Apple® IIc Technical Reference
Second Edition

Preface
Chapters 1 through 7

Simultaneously published in the United States and Canada.

Contents

8 Block Device I/O / 201

D Apple II–Family Differences / 455

Figures and tables

1 Introduction: The Apple IIc Family / 1

2 Address Map and Memory / 45

3 Resets and Interrupts / 77

4 Introduction to Apple IIc I/O / 107

7 Serial Port I/O / 171

8 Block Device I/O / 201

9 Mouse and Game Input / 251

Table 11-22 RAM address multiplexing / 343

Preface **About This Manual**

This is the reference manual for the Apple® IIc and Apple IIc Plus personal computers. It contains descriptions of all the hardware and firmware that are used in the Apple IIc family, and provides technical information that hardware designers and programmers need when developing products to be used with Apple IIc–family computers.

The information in this manual is aimed at assembly-language programmers and experienced hardware designers, but others interested in the internal operation of the Apple IIc and Apple IIc Plus can also benefit from reading it.

This preface describes the contents of this manual, and tells you how to obtain more technical information about Apple products. ■

This manual tells you how the Apple IIc–family members work, but not how to use them. If you need to know how to set up and use your Apple IIc or Apple IIc Plus, read the owner's manual that came with the computer.

This manual describes four versions of the Apple IIc:

- the original Apple IIc
- the first version of the Apple IIc that supports the UniDisk™ 3.5 drive
- the first version of the Apple IIc that supports the memory expansion card
- the Apple IIc Plus

More information on the various versions of the Apple IIc is provided in Chapter 1.

Contents of this manual

The following is a brief outline of the contents of this manual. See the table of contents for a complete list of the subjects covered in each chapter; refer to the index to locate a specific piece of information.

Chapter 1 introduces the Apple IIc family, including external controls, connectors, and the main internal components, and explains the differences between the various versions of the Apple IIc.

Chapter 2 describes the address space of the Apple IIc–family members, discusses the way each computer's memory is organized, and explains how to access the various areas of memory.

Chapter 3 discusses what happens when the Apple IIc or Apple IIc Plus is turned on or reset, and describes the use of interrupts in the Apple IIc family.

Chapter 4 introduces the characteristics and features of the standard I/O firmware routines for the Apple IIc family.

Chapter 5 describes how programs can read the keyboard and use the speaker.

Chapter 6 describes the firmware and memory features that allow your program to control the video display.

Chapter 7 describes the characteristics of the two serial ports and explains how your program can use these ports.

Chapter 8 discusses the firmware routines that communicate with block devices such as disk drives and memory expansion cards, and includes descriptions of all the SmartPort calls that you can use to control the disk-drive interface.

Chapter 9 describes the mouse port, including the different operating modes that you can select for the mouse, and how to read the mouse and hand controls.

Chapter 10 describes the Apple IIc family's built-in Monitor firmware. The Monitor helps you write, disassemble, and debug machine-language programs, as well as providing you with a means to look at and manipulate the contents of main memory.

Chapter 11 describes the Apple IIc–family hardware, including the power supply, all major components on the logic board, and all external and internal connectors.

Appendix A describes the 65C02 microprocessor in detail, including the differences between it and the 6502 microprocessor used on earlier models in the Apple II family. Most of this appendix is a reprint of the manufacturer's data sheet for the 65C02.

Appendix B contains an address map for the Apple IIc family, including detailed maps for memory pages $00 and $03, the screen holes, and the hardware page.

Appendix C describes some of the operating systems and languages supported by Apple Computer for the Apple IIc family.

Appendix D outlines the differences and similarities between the different members of the Apple II family of computers.

Appendix E describes the various international versions of the Apple IIc keyboard and character set. Power and safety information for international versions of the Apple IIc are also included in this appendix.

Appendix F lists the firmware entry points for the Apple IIc family ROM that are supported by Apple Computer, Inc. A description of the purpose and use of each entry point is included.

Appendix G contains some tables that you might find useful, such as an ASCII table with binary, hexadecimal, and decimal equivalents, and hexadecimal to decimal conversion tables.

The glossary defines many of the technical terms used in this manual.

Conventions used in this manual

This section explains the typographic and terminology conventions used in this manual.

Typographic conventions

Special text in this manual is set off in several different ways, as shown in these examples.

▲ **Warning** Warnings like this direct your attention to something that could cause injury, damage either software or hardware, or result in loss of data. ▲

△ Important Text set off like this contains information that is especially important. △

△ Original IIc Text set off like this—with the names of one or more computers in the left margin—applies only to the specified computer or computers. △

◆ *By the way:* Information that is useful but incidental to the text is set off like this. You may want to skip over such information or return to it later.

Specialized terms are printed in **boldface** when they are first defined; these terms are defined in the glossary as well.

`Computer voice` is used to indicate text that should be identical to your screen display or printout.

Notation and terminology conventions

A dollar sign ($) before a number indicates that the number is in hexadecimal notation. For example, the hexadecimal equivalent of decimal 16 is written as $10.

An asterisk (*) after a signal name indicates that the signal is active low—that is, it is "on" when the signal is at a TTL low (0V) level

The following abbreviations are used:

KB	kilobyte: 1,024 bytes
Kbit	kilobit: 1,024 bits
MB	megabyte: 1,024 kilobytes, or 1,048,576 bytes

This book distinguishes between *boards* and *cards* as follows: a board is a permanent part of the computer (for example, the main logic board), whereas a card can be added or exchanged by the user to expand or reconfigure the system.

The two special function keys now called *Command* and *Option* have had other names in the past. The key with the outline of an apple on the keycap was originally called the *Open Apple key*, then the *Apple key*, and is now called the *Command key*. The versions of the Apple IIc before the Apple IIc Plus have a key with a solid apple on the keycap; this key was originally called the *Solid Apple Key*, but is now called the Option key. On the Apple IIc Plus, this key has *Option* on the keycap; there is no solid apple symbol on the keyboard. In this book, the names *Command* and *Option* are always used to refer to these two keys.

How to get more information

Several organizations exist that provide support for Apple II programmers and users. This section tells you how to contact the Apple Programmer's and Developer's Association (APDA™), Apple user groups, and Apple Developer Services.

APDA

APDA provides a wide range of technical products and documentation, from Apple and other suppliers, for programmers and developers who work on Apple equipment. For information about APDA, contact

APDA
Apple Computer, Inc.
20525 Mariani Avenue, Mailstop 33-G
Cupertino, CA 95014-6299

800-282-APDA (800-282-2732)
Fax: 408-562-3971
Telex: 171-576
AppleLink: APDA

User groups

Ask your authorized Apple dealer for the name of the Apple user group nearest you, or call 800-538-9696. For information about starting your own user group, contact

The Boston Computer Society
One Center Plaza
Boston, MA 02108
USA
617-367-8080

Apple Developer Services

The Apple Certified Developer program is for professional hardware and software developers who plan to have a finished commercial product within 18 months. As a Certified Developer, you will receive monthly mailings including a newsletter, Apple II and Macintosh Technical Notes, pertinent Developer Program information, and all the latest news relating to Apple products. You will have access to Apple's Developer Hotline for general developer information.

Once you are certified, Apple's Developer Technical Support staff can provide you with technical assistance. Apple's Technical Support engineers answer questions within 24 hours by electronic mail.

For information about getting certified as an Apple developer, contact Apple Developer Services at the following address:

Apple Developer Services
Apple Computer, Inc.
20525 Mariani Avenue, Mailstop 27-S
Cupertino, California 95014-6299

You will have to submit information on previous products and your present business plan along with your completed application.

Chapter 1 **Introduction: The Apple IIc Family**

Several enhancements have been made to the Apple® IIc computer since the original version was introduced in 1984. This chapter introduces the features of the different versions of the Apple IIc computer, including the latest member of the Apple IIc family, the Apple IIc Plus. ■

The first enhancement to the Apple IIc provided support for the UniDisk™ 3.5 external drive, and included a set of ROM-based assembly-language routines called the **Protocol Converter.** The second enhancement of the Apple IIc included a new version of the Protocol Converter called the **SmartPort,** and support for an optional memory expansion card. The third enchancement, the Apple IIc Plus computer, has a faster microprocessor than that used in earlier Apple IIc computers, a 3.5-inch internal disk drive rather than the 5.25-inch internal disk drive used formerly, an internal power supply rather than the separate, external power supply used formerly, and a new keyboard. All versions of the Apple IIc are described in this manual.

◆ *Note:* The Protocol Converter and SmartPort are different names for essentially the same built-in program that handles input/output operations to peripheral devices like disk drives, hard disks, and tape backup machines. SmartPort is described in detail in Chapter 8.

Where there are differences between the various versions of the Apple IIc, they are called out in the manual. For the sake of convenience, the first two enhanced versions of the Apple IIc are identified by the new features they support: the *UniDisk 3.5 Apple IIc* and the *memory expansion Apple IIc*. Unless otherwise specified, all versions of the Apple IIc operate identically.

Software identification of an Apple IIc

An Apple II program can tell that it is running on an Apple IIc by determining that the identification byte in location $FBB3 of ROM is equal to $06 and that the byte in location $FBC0 is $00. A program can tell which member of the Apple IIc family it is running on by checking the value of the identification byte in location $FBBF of ROM. Table 1-1 lists the versions of the Apple IIc and their identification bytes. For the identification bytes to distinguish between all members of the Apple II family, see Table D-1 in Appendix D.

■ **Table 1-1** Apple IIc–family identification bytes

Machine	$FBBF
Apple IIc (original)	$FF
Apple IIc (UniDisk 3.5)	$00
Apple IIc (memory expansion)	$03
Apple IIc (memory expansion, revised ROM)	$04
Apple IIc Plus	$05

◆ *Checking the ID byte:* You can use the PEEK command in Applesoft BASIC to check the values of the ID bytes. For example, to get the value of the ID byte at $FBBF (64447 decimal), type `PRINT PEEK (64447)`.

Versions of the Apple IIc ROM

There have been five versions of the Apple IIc ROM so far. The ROM for the original Apple IIc assigned I/O devices to ports 1 through 6 (see the section "Port I/O," in Chapter 4); the mouse firmware entry points are assigned to port 4. A PR#7 command from Applesoft BASIC causes the original Apple IIc to boot from the external 5.25-inch disk drive. The serial firmware in the original Apple IIc does not support AppleTalk® or XON/XOFF protocol, and does not mask incoming line-feed characters. The original Apple IIc can use 5.25-inch floppy disk drives only.

The firmware in the UniDisk 3.5 Apple IIc is functionally identical to that in the original Apple IIc except that the UniDisk 3.5 firmware

- supports UniDisk 3.5 disk drives in addition to 5.25-inch disk drives
- has AppleTalk firmware in port 7
- returns the message "AppleTalk Off Line" in response to a PR#7 command

- masks all incoming line-feed characters as a default setting
- supports XON/XOFF protocol

The firmware in the memory expansion Apple IIc is functionally identical to that in the UniDisk 3.5 Apple IIc except that the memory expansion firmware

- supports an internal memory expansion (RAM disk) card assigned to port 4
- assigns the mouse to port 7
- has no AppleTalk firmware
- cannot respond to a PR#7 command (if you try to execute a PR#7 command, the system hangs)

The firmware in the revised memory expansion Apple IIc is functionally identical to that in the earlier memory expansion Apple IIc except that the revised firmware corrects a couple of minor bugs in the earlier firmware.

The firmware in the Apple IIc Plus is functionally identical to that in the memory expansion Apple IIc except that the Apple IIc Plus firmware

- supports an internal 3.5-inch disk drive and external Apple 3.5-inch disk drives in addition to UniDisk 3.5 disk drives and 5.25-inch disk drives
- supports the cache glue gate array (CGGA) and other hardware that enables the CPU to operate at up to 4 MHz

Other differences between the various versions of the Apple IIc family computers are described in the following sections.

The original Apple IIc

The original Apple IIc is the first member of the IIc family. It has the following features:

- a 65C02 microprocessor
- 128 KB of RAM
- a 5.25-inch internal disk drive

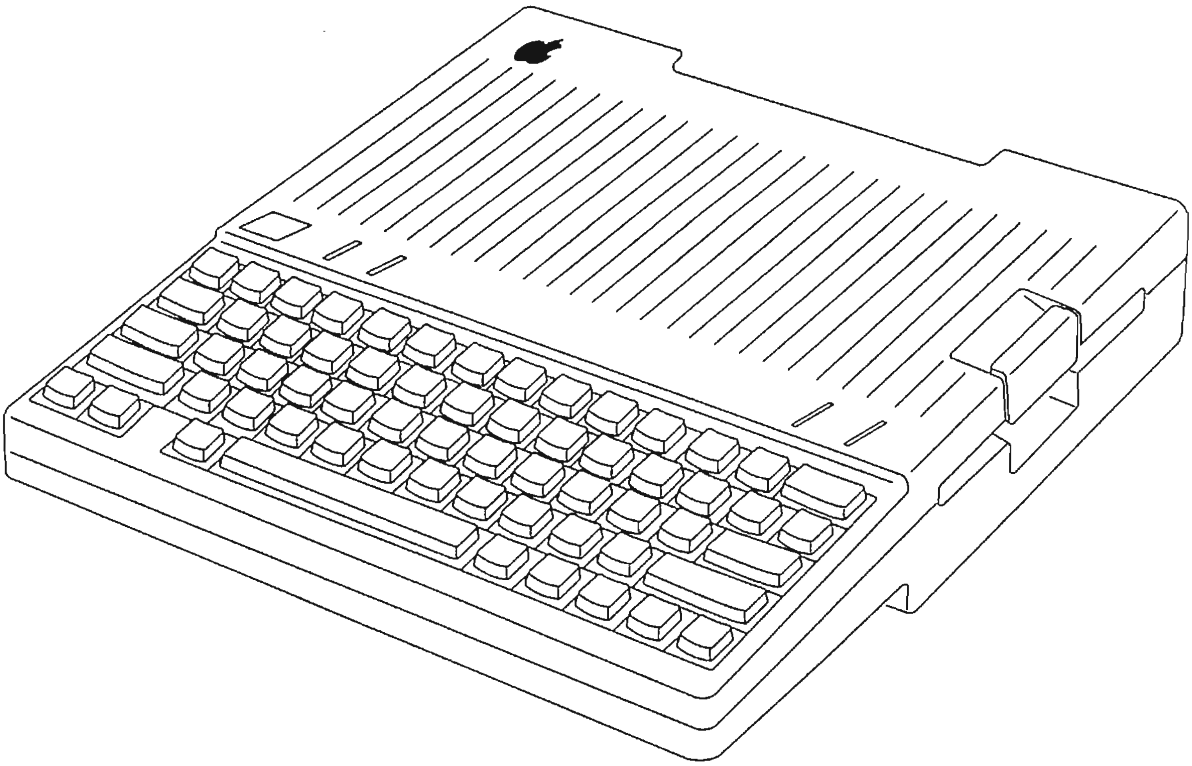
The outside of the original Apple IIc

This section briefly describes the original Apple IIc keyboard, controls, indicators, and peripheral device connectors.

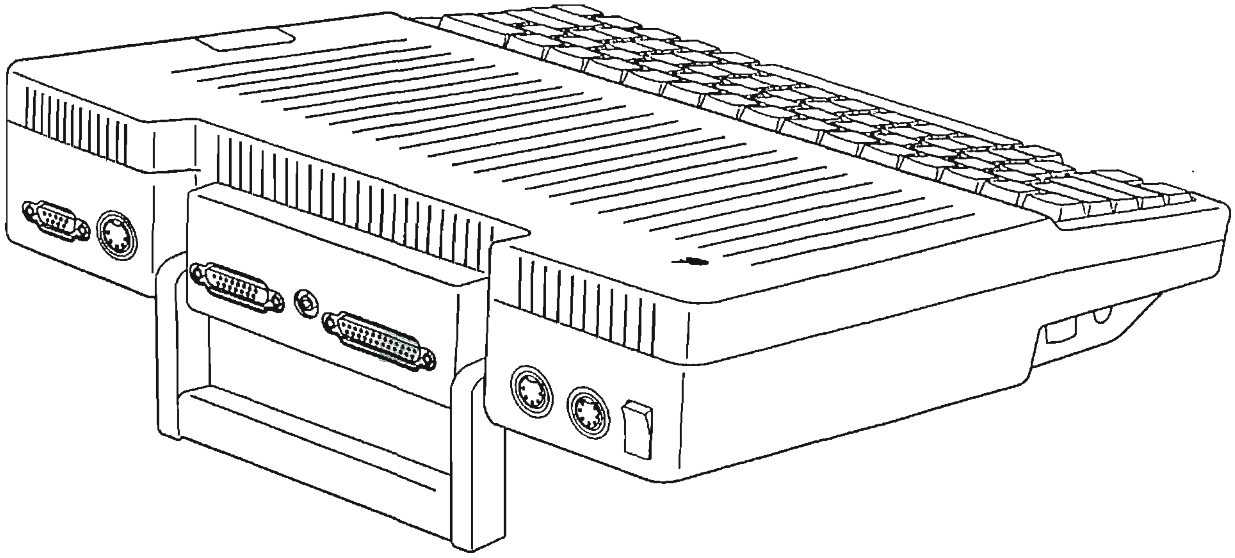
The original Apple IIc comes equipped with a keyboard, speaker (with audio output jack and volume control), internal disk drive, external power supply, and internal voltage converter. It also has built-in interfaces for a serial printer, a video monitor, special video display adapters, a modem, a mouse, and game controllers. These interfaces allow you to connect peripheral devices without having to go inside the machine to use expansion slots.

Figure 1-1 shows the front and right side of an original Apple IIc, and Figure 1-2 shows the back and left side.

■ **Figure 1-1** Original Apple IIc external features, front



■ **Figure 1-2** Original Apple IIc external features, back



The keyboard

The primary input device of the original Apple is the keyboard, shown in Figure 1-3. The keyboard has a 62-key typewriter layout with both uppercase and lowercase characters and can generate all 128 standard ASCII characters. A reset switch, 80/40-column display selector switch, keyboard layout selector switch, disk-use light, and power light are also located on the front of the computer.

◆ *Note:* **ASCII** stands for ***American Standard Code for Information Interchange.*** It is a standard for assigning data values to letters, symbols, and numbers (that is, "encoding" them) so that a computer can handle them. Table 5-2 lists the ASCII encoding for the standard and simplified USA keyboard characters. Appendix E lists the encoding for international keyboards.

■ **Figure 1-3** Front of original Apple IIc with standard USA keyboard

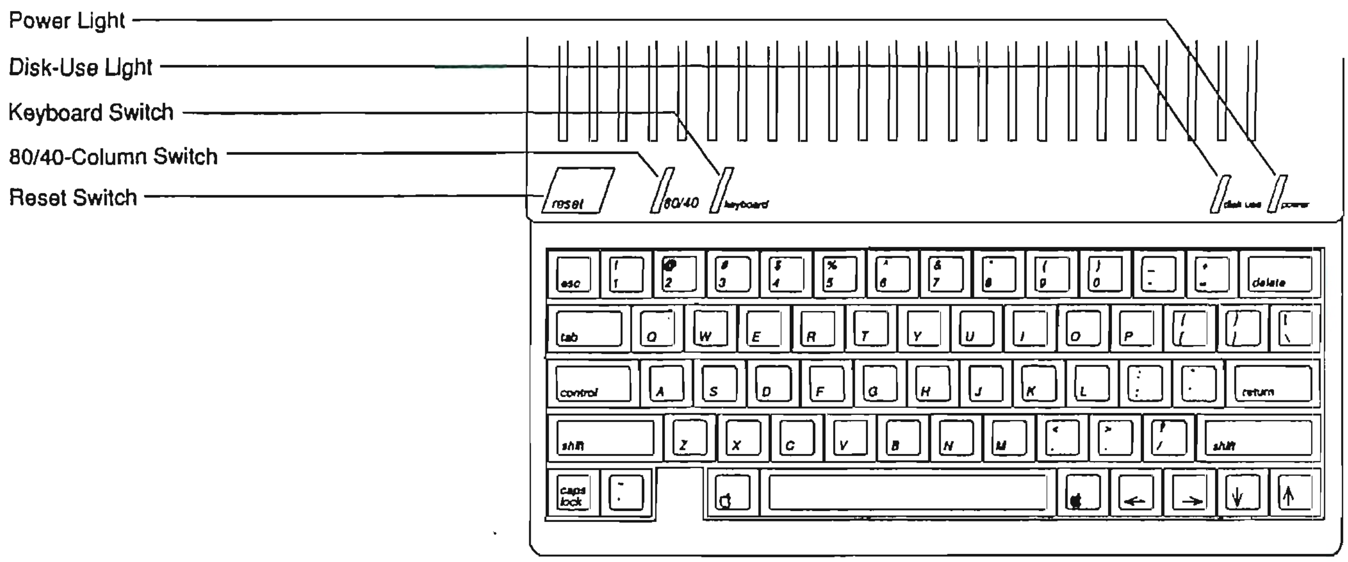


Table 1-2 lists the characteristics of all original Apple IIc keyboards and front panels.

Operating features

The original Apple IIc keyboard has automatic repeat on all character keys. This means that if you hold the key down longer than about a second, the character it generates repeats until you let up the key. It also has two-key rollover, which means if you press a key before releasing the one you pressed before it, the second character enters the computer as though you had released the previous key.

■ Table 1-2 Original Apple IIc keyboard specifications

Number of keys	62
Character encoding	ASCII
Number of codes	128
Features	Automatic repeat, two-key rollover
Cursor movement keys	Left Arrow, Right Arrow, Down Arrow, Up Arrow, Return, Delete, Tab
Modifier keys	Control, Shift, Caps Lock
Special function keys	Command (Open Apple), Option (Solid Apple), Esc (Escape)
Front-panel switches	Reset, 80/40 switch, keyboard switch
Front-panel lights	Power light, disk-use light

Cursor movement keys

The original Apple IIc keyboard has four cursor movement keys with arrows marked on them: left, right, down, and up. Three other keys can also cause cursor movements: Return, Delete, and Tab. All seven of these keys generate ASCII control characters. It is up to the operating system or application program to interpret and act on the ASCII codes that these keys generate. The control characters and their corresponding control codes are shown in Table 5-2.

Modifier keys

Three special keys—Control, Shift, and Caps Lock—generate no codes when pressed by themselves, but change the codes generated by other keys with which they are pressed.

- Control, when pressed in combination with letter keys or certain other keys, produces ASCII control characters. Most applications do not display characters on the screen to correspond to control characters.
- Shift works the same on the original Apple IIc as on an ordinary typewriter: it selects uppercase letters and the upper characters on nonletter keys.
- Caps Lock, in its down position, changes the letter keys to uppercase, but does not affect other keys.

Special function keys

The original Apple IIc keyboard has three special function keys: Command, Option, and Esc (Escape).

The Command key is marked with the outline of an apple. The Option key is marked with a filled-in apple. The Command and Option keys do not have ASCII values; instead, they affect the values of fixed 1-bit locations in main memory. These addresses and their use are described in Chapters 5 and 9.

The Esc key generates the ASCII escape (ESC) control character (key code $1B—see Table 5-2). This is the same character as generated by Control–left bracket (Control-[).

When the Esc key is pressed, many programs—including the System Monitor in the Apple IIc ROM—then interpret the keystrokes that follow as an **escape sequence.**

Front panel switches

Above the keyboard of the original Apple IIc are three switches: Reset, the 80/40-column switch, and the keyboard layout switch.

The Reset key has a direct line to the 65C02 microprocessor RESET signal line: holding down Control while pressing Reset causes the original Apple IIc to execute a warm start. See Chapter 3 for more details on resets and warm starts.

You can restart the original Apple IIc without turning the power off and back on again by holding down both Control and Command while pressing Reset. This procedure causes the original Apple IIc to execute a forced cold reset. A forced cold reset puts the Apple IIc into the same state as if you had turned the power off and back on, without putting as much strain on the power supply as does the power-down/power-up sequence.

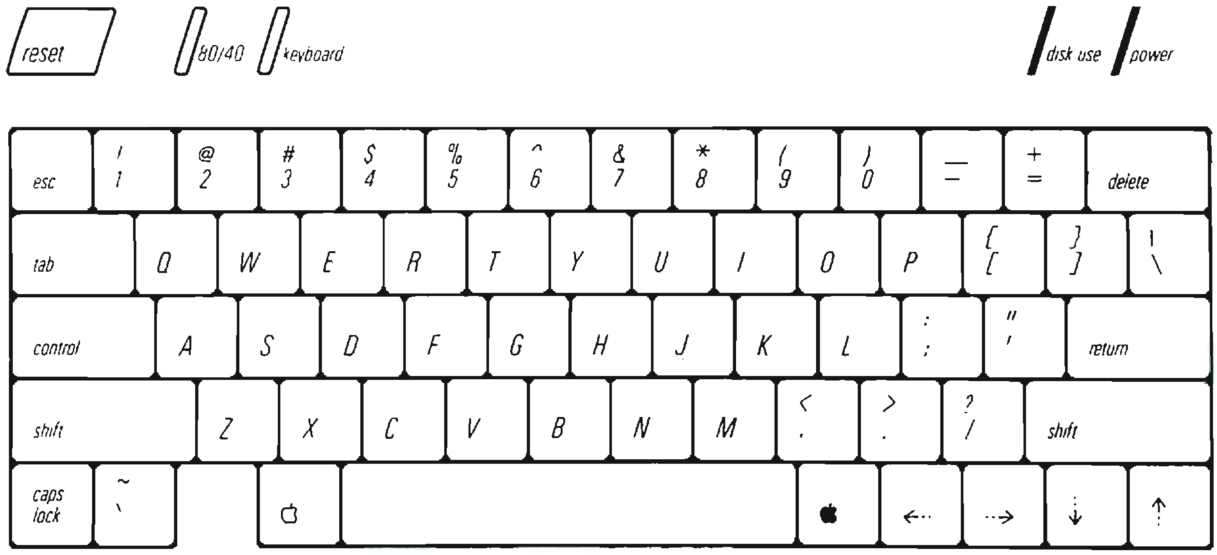
The 80/40-column switch on the keyboard of the original Apple IIc lets you specify whether a program should display information in 40 or 80 columns per line. The switch indicates 40-column display when in its down position, and 80-column display when in its up position.

△ Important Not all programs check the 80/40 switch. Even programs that do check the switch may do so only when the program first starts up. If that is the case, changing the switch position while the program is running has no effect on the program's display. (See Table 5-1.) △

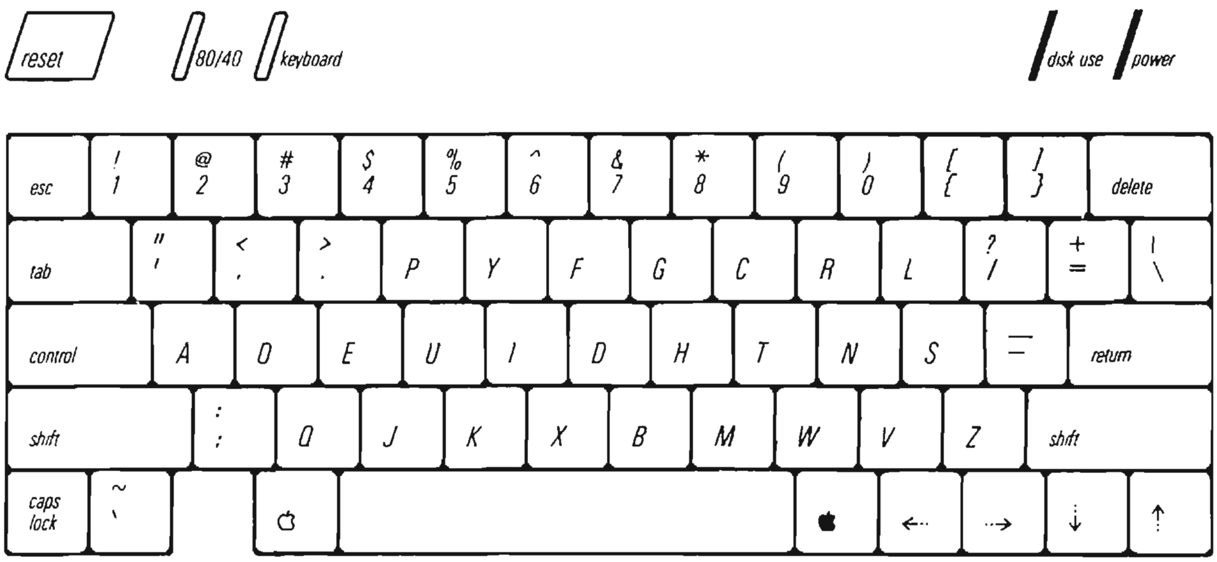
You use the keyboard layout switch to select which of the two keyboard layouts and screen character sets built into your original Apple IIc you want to use. On USA versions of the original Apple IIc, you select the standard keyboard layout (known as the **QWERTY** or *Sholes* layout) (Figure 1-4) with the switch in the up position, and the **Dvorak** simplified layout (Figure 1-5) with the switch in the down position.

The Apple IIc comes from the factory with the labels on the keycaps arranged in the Sholes positions. If you normally use the Dvorak keyboard layout and wish to change the keycaps, *gently* pry up the keycaps from the keyboard and rearrange and replace them in their Dvorak positions.

■ **Figure 1-4** Key assignments for original Apple IIc keyboard, with keyboard switch up (standard Sholes position)



■ **Figure 1-5** Key assignments for original Apple IIc keyboard, with keyboard switch down (Dvorak position); shaded characters may be in different positions on some models.



On international models, the labels on the keycaps indicate the character positions for the local keyboard layout, which is selected when the keyboard switch is down. When up, the keyboard switch selects the USA standard characters and key layout. The keyboard layouts and character sets for USA and international models of the original Apple IIc are provided in Appendix E.

Disk-use and power lights

The red disk-use light glows whenever the internal disk drive's motor is switched on.

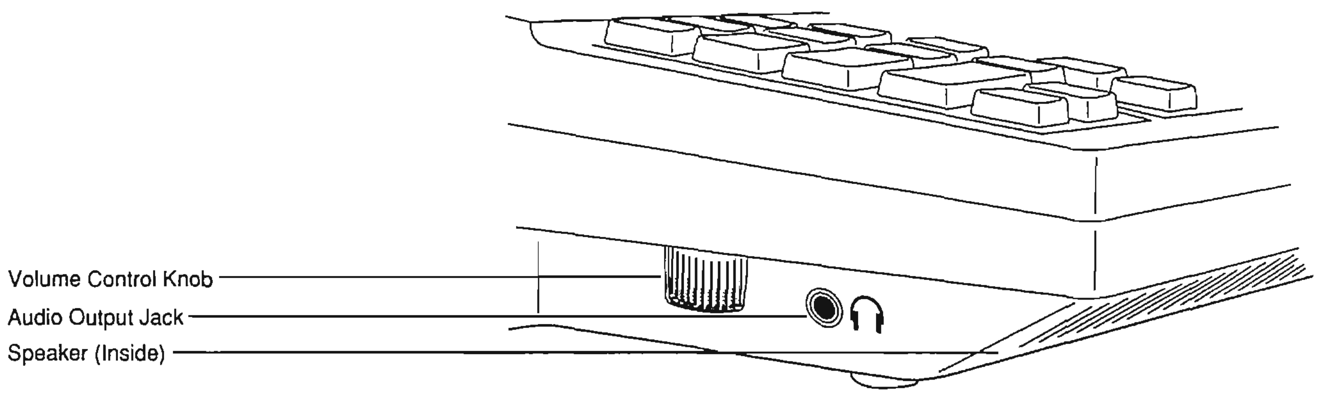
The green power light glows when the original Apple IIc is turned on and normal power is present at the internal voltage converter of the original Apple IIc.

▲ **Warning** If the power light flashes on and off, turn off the computer immediately. Find out what caused the condition (such as a brownout or short circuit) and fix the problem before turning the computer on again. Above all, do not use the disk drive when the power light is flashing; doing so may damage the computer. ▲

The volume control and audio output jack

The original Apple IIc has a built-in speaker that lets application programs produce a variety of sounds. There is a volume control on the left side of the original Apple IIc case, and a jack for connecting headphones or an external speaker, as shown in Figure 1-6. The jack accepts either one-channel (monaural) or two-channel (stereo) plugs, although speaker output is monaural only. Inserting a plug disconnects the built-in speaker. Use of the speaker by application programs is explained in Chapter 5.

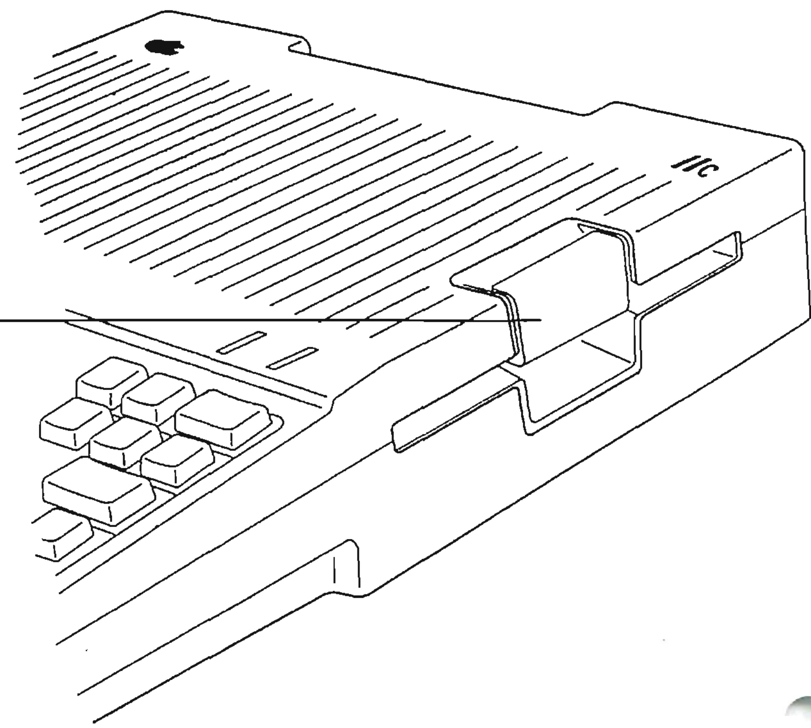
■ **Figure 1-6** Apple IIc volume control and audio output jack



The door for the internal disk drive

Disks are inserted and removed from the original Apple IIc's internal disk drive via a door in the right side of the case (Figure 1-7).

■ **Figure 1-7** Internal disk drive door



The back panel

The back panel of the original Apple IIc (Figure 1-8) has seven connectors and a main power switch. From left to right they are

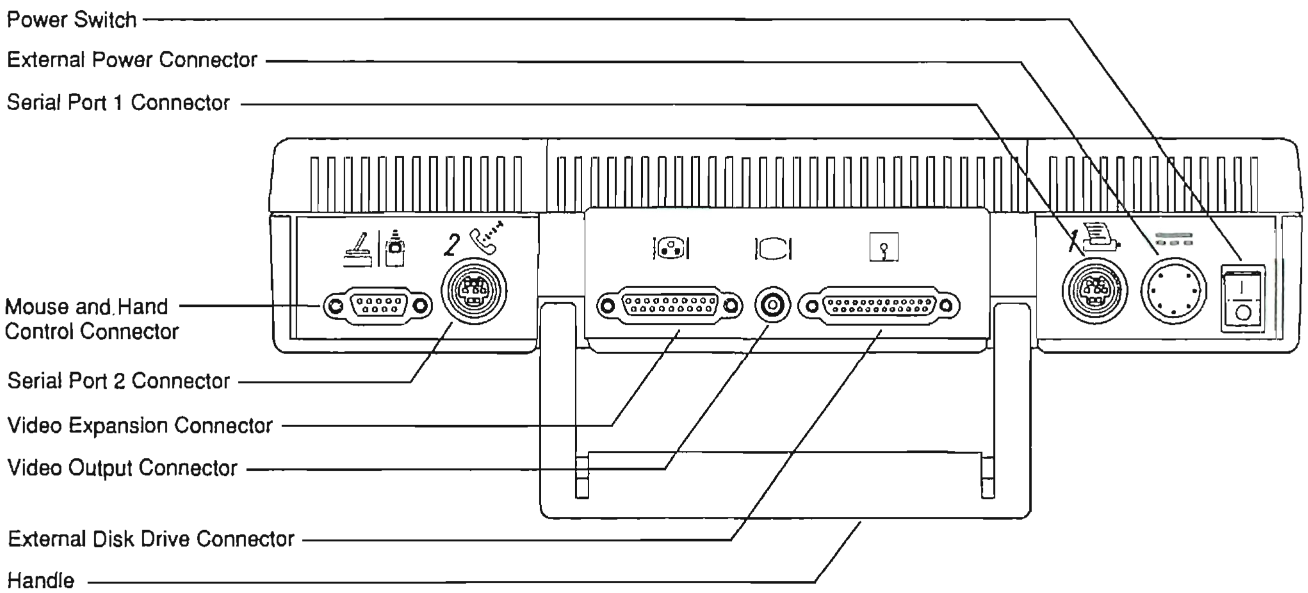
- a 9-pin D-type (DB-9) miniature connector for connecting hand controllers, a mouse, a joystick, or some other device (see Chapters 9 and 11)
- a 5-pin DIN connector for serial input and output (port 2, normally for a modem) (see Chapters 7 and 11)
- a 15-pin D-type (DB-15) connector for video expansion (see Chapter 11)
- an RCA-type jack for a video monitor (see Chapter 11)
- a 19-pin D-type (DB-19) connector for connecting one or more external devices, such as intelligent disk drives (see Chapters 8 and 11)

- another 5-pin DIN connector for serial input and output (port 1, normally for a printer or plotter) (see Chapters 7 and 11)
- a special 7-pin DIN connector for power input (see Chapter 11)

Before attaching cables to the original Apple IIc back panel connectors, be sure to move the handle until it clicks into position for propping up the computer. The handle should be down whenever the computer is running so that the computer can maintain proper cooling airflow.

The installation manuals for external devices contain instructions for connecting them to the Apple IIc.

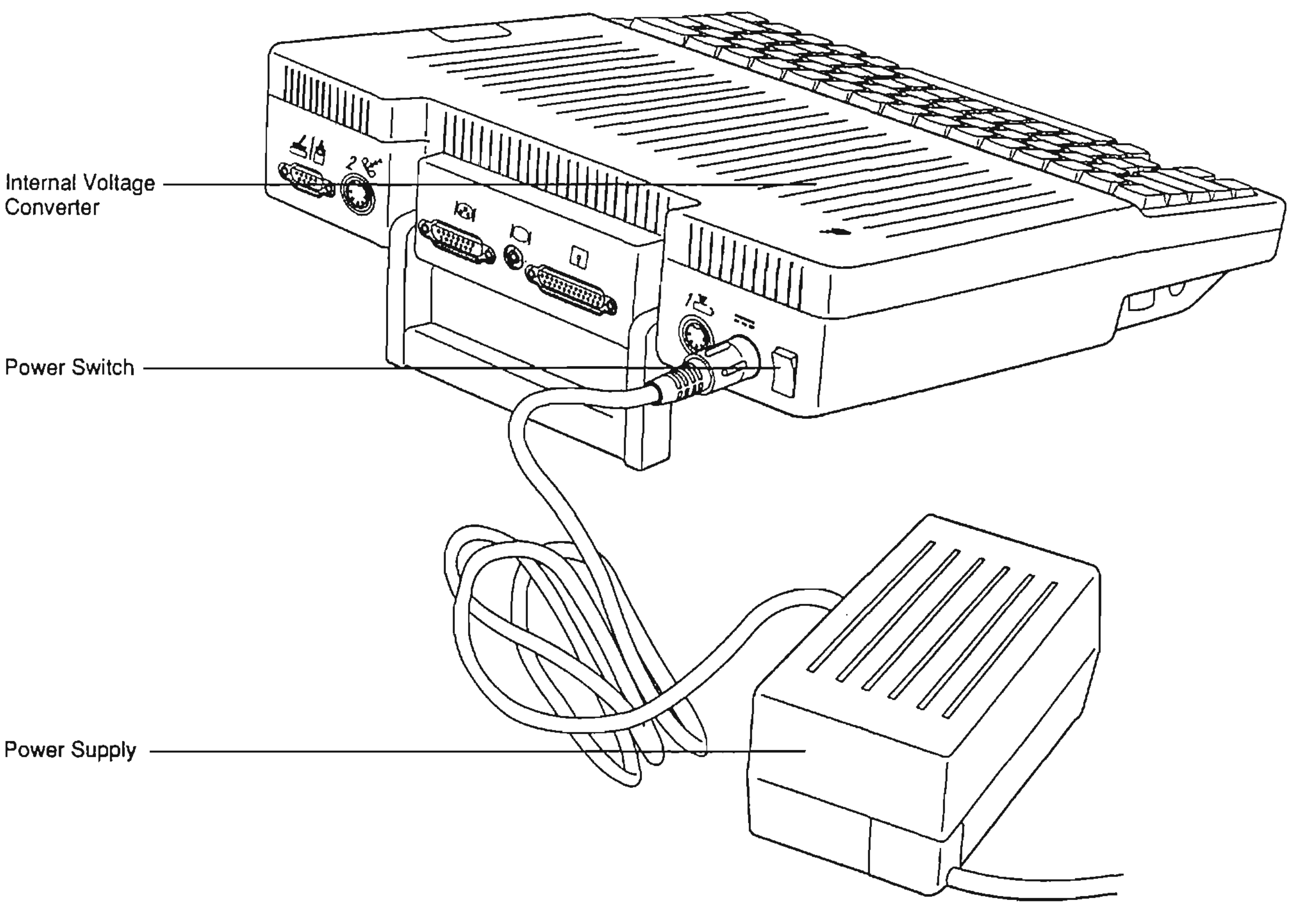
■ **Figure 1-8** Original Apple IIc back panel connectors



The external power supply

The original Apple IIc computer comes with a separate external power supply that plugs into a 115-volt source (see Figure 1-9). The power supply provides 1.2 amps of current at +15 volts to the original Apple IIc. The internal voltage converter then converts the power to the voltages needed within the machine. The external power supply is described in more detail in Chapter 11.

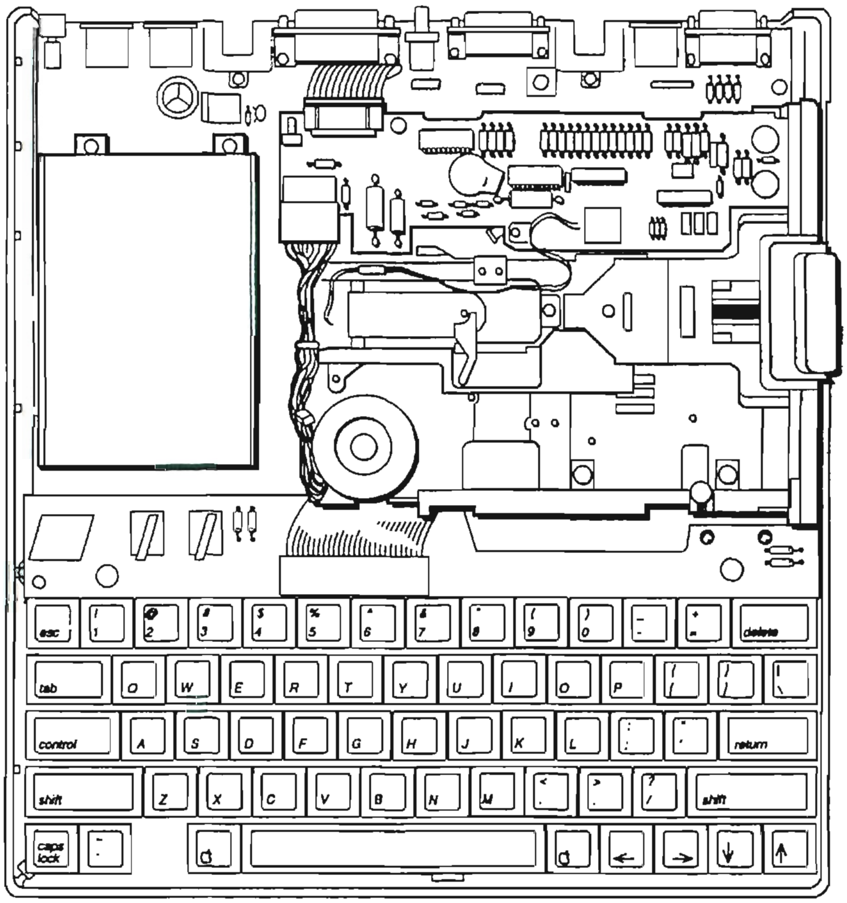
■ **Figure 1-9** Power supply and voltage converter



The inside of the original Apple IIc

Figure 1-10 shows the main components inside the original Apple IIc computer.

■ **Figure 1-10** Inside the original Apple IIc



The internal voltage converter

The built-in voltage converter in the original Apple IIc takes a +15 VDC (nominal) input source and produces three different voltages: +5 volts, +12 volts, and -12volts. The input source is normally the external power supply furnished with the original Apple IIc (Figure 1-9). The voltage converter provides power for the main logic board, the internal disk drive, one external disk drive, and the I/O signals available at the back panel.

The voltage converter is a high-efficiency switching converter that protects itself and the rest of the original Apple IIc against short circuits and other electrical mishaps.

The main logic board

The original Apple IIc main logic board, which is mounted flat in the bottom of the case, contains almost all the electronic parts of the computer.

Figure 1-11 shows the main logic board and the most important integrated circuits (ICs) in the original Apple IIc. They are

- the CPU (central processing unit)
- the RAM (random-access memory)
- the ROM (read-only memory)
- the five custom integrated circuits (ICs), or ASICs (application-specific integrated circuits)

The CPU is a 65C02 microprocessor. The 65C02 is a CMOS version of the 6502, the microprocessor used in earlier members of the Apple II family. The 65C02 is an 8-bit microprocessor with a 16-bit address bus. In the original Apple IIc, the 65C02 runs at 1 MHz and performs up to 500,000 8-bit operations per second.

The RAM provides temporary storage for information to which the CPU must have access, such as application programs or data being manipulated by an application program.

The ROM provides permanent storage for information to which the CPU or other ICs must have access. There are three ROMs in the original Apple IIc computer: the keyboard ROM, the character generator ROM, and the system ROM. The keyboard ROM translates the key codes coming from the keyboard into ASCII codes. The character generator ROM converts ASCII character values to a form that the video display can use. The system ROM contains the Monitor, the Applesoft BASIC interpreter, enhanced video firmware, and other input/output firmware. The keyboard and character generator ROMs are discussed in Chapter 11. The routines in system ROM are discussed throughout this book. The Applesoft language interpreter is described in the *Applesoft Tutorial* and the *Applesoft BASIC Programmer's Reference Manual.*

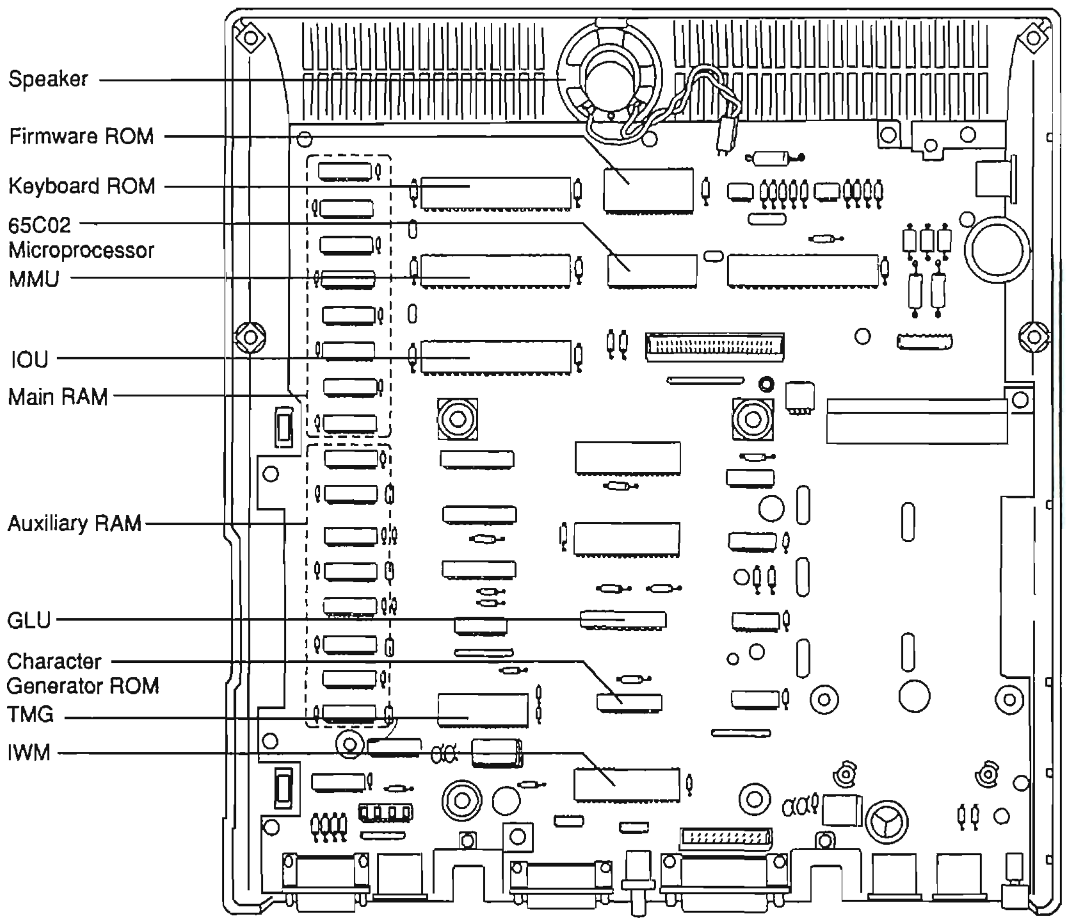
Five of the large ICs on the main logic board of the original Apple IIc are custom-made for the Apple IIc computer:

- The **memory management unit** (MMU) contains most of the logic that controls memory addressing in the original Apple IIc.
- The **input/output unit** (IOU) contains most of the logic that controls the built-in input and output features of the original Apple IIc.
- The **timing generator** (TMG) uses the signal from a 14 MHz oscillator to generate all the clock and timing signals used by the system.
- The **general logic unit** (GLU) performs a variety of logic functions.

- The **disk controller unit**, also known as the **Integrated Woz Machine** (IWM), is a single-chip version of the Apple Disk II controller card. It controls the built-in and external disk drives connected to the original Apple IIc.

These components are discussed more fully in Chapter 11.

- **Figure 1-11** Original Apple IIc main logic board



The other circuit boards

The original Apple IIc contains two other circuit boards that serve special purposes: a motor-speed control and read/write logic board for the disk drive, and a matrix board for detecting the position of keys pressed. This manual does not discuss these circuit boards.

▲ **Warning** Adjustment of disk drive speed must be done by an authorized Apple Service Center. Do not attempt to adjust the speed of your internal disk drive. If you do, you may damage the disk drive and you will void your warranty. ▲

The UniDisk 3.5 Apple IIc

The Apple IIc that introduced support for the UniDisk 3.5 drive is referred to in this manual as the *UniDisk 3.5* version. It includes the following changes from the original Apple IIc:

- the addition of a set of routines in ROM, known as the *Protocol Converter*, which support the UniDisk 3.5 external disk drive
- the replacement of the 16 KB ROM IC with a 32 KB ROM IC
- the addition of some new serial port commands
- the addition of the mini-assembler, a routine in ROM that you can use to write machine-language routines for the Apple IIc
- the addition of two new Monitor commands (Step and Trace)
- the addition of built-in diagnostics routines

The UniDisk 3.5 Apple IIc also includes an improved interrupt handler and new external-drive startup procedures.

The outside of the UniDisk 3.5 Apple IIc

The outside of the UniDisk 3.5 Apple IIc is identical to that of the original Apple IIc. A UniDisk 3.5-inch disk drive connects to the external disk drive port just like a Disk II 5.25-inch drive does.

The inside of the UniDisk 3.5 Apple IIc

The inside of the UniDisk 3.5 Apple IIc is almost exactly the same as that of the original Apple IIc, with one exception: the new system ROM. This 32 KB ROM IC holds twice as much information as the 16 KB ROM used in the original Apple IIc. Chapter 11 contains the pinout diagram for this IC.

The memory expansion Apple IIc

The first Apple IIc that supports an optional memory expansion card is referred to in this manual as the *memory expansion version*. It includes the following changes from the UniDisk 3.5 version:

- the addition of an internal connector to support an optional memory expansion card
- the replacement of the sixteen 8 KB RAM ICs with four 32 KB RAM ICs

The memory expansion version also includes some firmware changes: the mouse, located at port 4 in the original and UniDisk 3.5 versions, is at port 7 in the memory expansion version. A memory expansion card uses port 4 in the memory expansion Apple IIc. In other words, all the mouse I/O entry point addresses have been changed from $C4xx to $C7xx.

◆ *Note:* Both the memory expansion Apple IIc and the Apple IIc Plus computer have built-in firmware and connectors to support memory expansion cards. Apple Computer, Inc., no longer manufactures a memory expansion card for the memory expansion Apple IIc, and has never manufactured a memory expansion card for use in the Apple IIc Plus computer. Although the memory expansion card connector in the Apple IIc Plus is identical to that in the memory expansion Apple IIc, the Apple Memory Expansion Card does not function in the Apple IIc Plus.

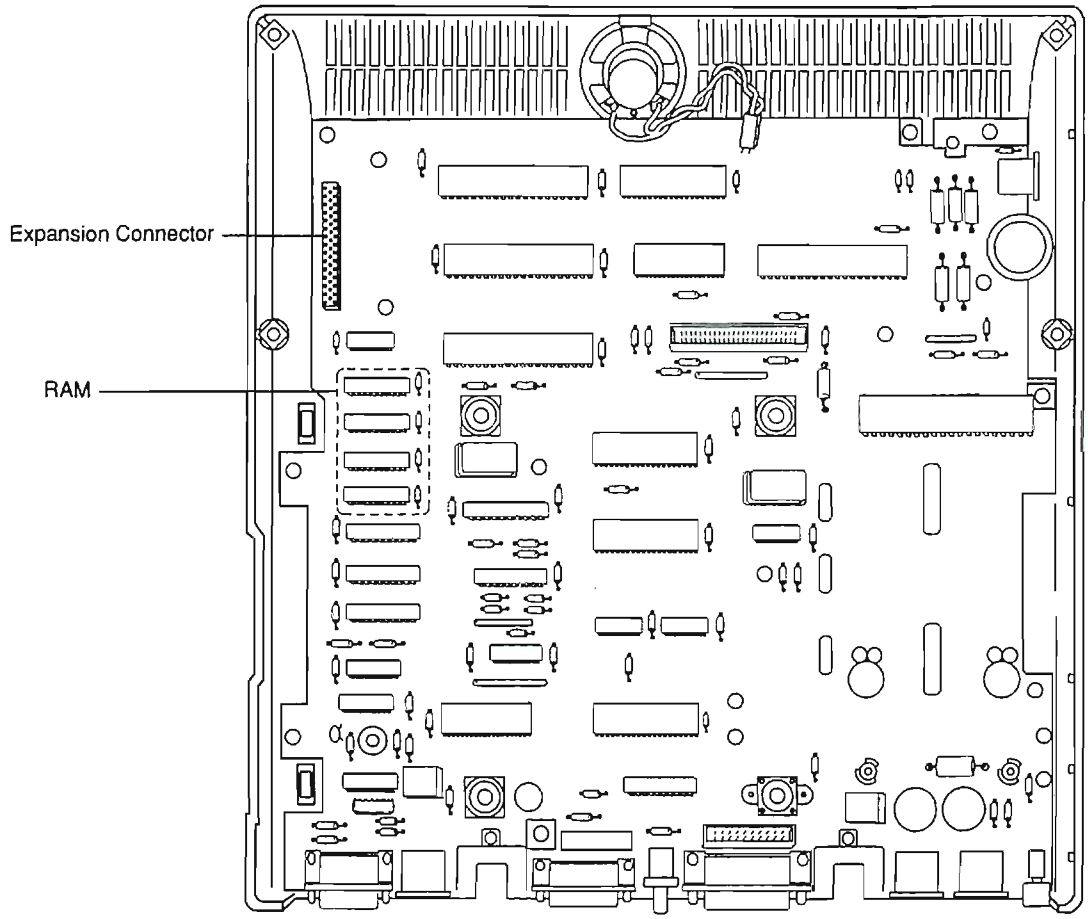
The outside of the memory expansion Apple IIc

The outside of the memory expansion Apple IIc is identical to that of the original Apple IIc.

The inside of the memory expansion Apple IIc

The inside of the memory expansion Apple IIc differs from that of the original and UniDisk 3.5 versions only in that the main logic board uses four 32 KB RAM ICs to replace the sixteen 8 KB ICs used in the earlier versions of the Apple IIc and that the main logic board in the memory expansion version has a 34-pin right-angle connector to connect the optional memory expansion card. Figure 1-12 shows the main logic board with the 8 KB RAM ICs and the memory expansion card connector.

■ **Figure 1-12** Memory expansion Apple IIc main logic board



The Apple IIc Plus

The newest Apple IIc is called the Apple IIc Plus computer. The Apple IIc Plus includes the following changes from the memory expansion Apple IIc:

- the replacement of the 1 MHz 65C02 microprocessor with a 4 MHz version of the same chip
- the addition of a 16 MHz crystal oscillator, an 84-pin custom IC, and two 8 KB static RAMs (together with the new microprocessor, these components permit the Apple IIc Plus to operate much faster than other members of the Apple IIc family)

- the replacement of the 5.25-inch internal disk drive with a 3.5-inch internal disk drive
- the addition of an ASIC and a 2 KB static RAM IC that provide logic for the internal and external 3.5-inch drives
- the replacement of the keyboard with one that uses the Apple Standard Keyboard layout
- the replacement of the separate external power supply and internal voltage converter with a completely internal power supply
- the replacement of the 5-pin DIN serial port connectors with 8-pin mini-DIN connectors
- the deletion of the audio output jack
- the replacement of the volume control knob with a sliding control located above the keyboard
- the deletion of the 40/80 column switch; 40/80 column switching is now software controlled
- the replacement of the sound hybrid chip with an equivalent circuit

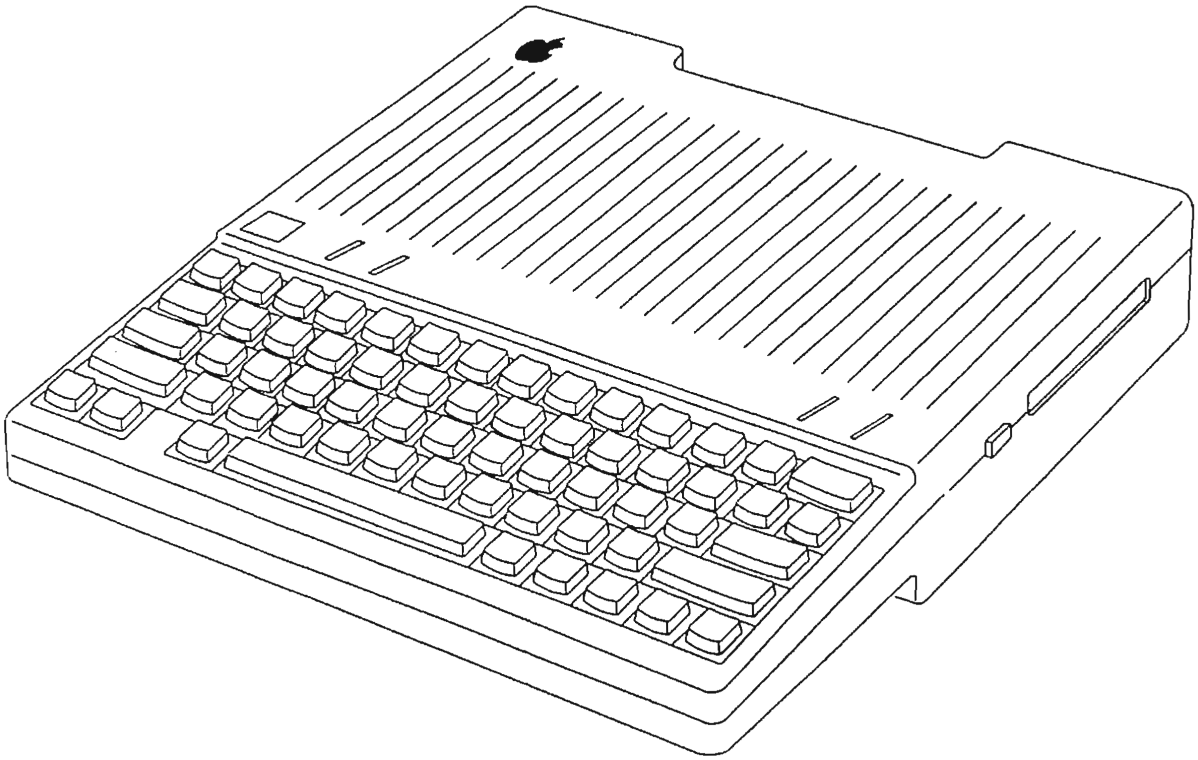
◆ *Note:* Both the memory expansion Apple IIc and the Apple IIc Plus have built-in firmware to support memory expansion cards. Apple Computer, Inc., no longer manufactures a memory expansion card for the memory expansion Apple IIc, and has never manufactured a memory expansion card for use in the Apple IIc Plus computer. The memory expansion card that was manufactured by Apple Computer, Inc., for use in the memory expansion Apple IIc does not work in the Apple IIc Plus computer. To add a memory expansion card to the Apple IIc Plus, you must purchase a card made by another manufacturer specifically for the Apple IIc Plus.

The outside of the Apple IIc Plus

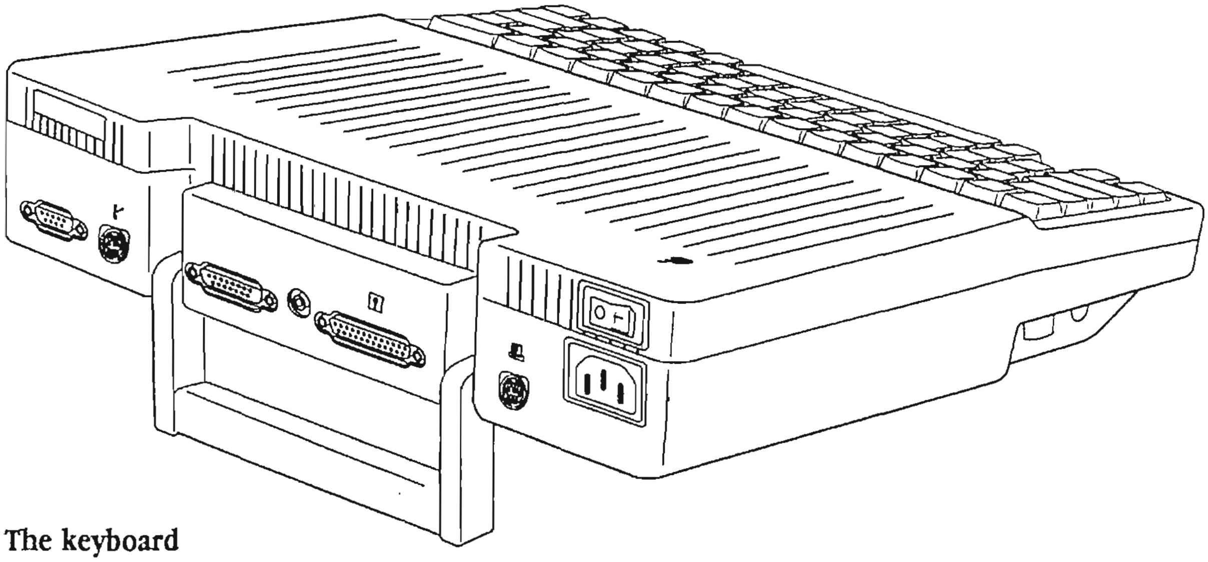
This section briefly describes the Apple IIc Plus keyboard, controls, indicators, and peripheral-device connectors.

Figure 1-13 shows the front and right side of the Apple IIc Plus, and Figure 1-14 shows the back and left side.

■ **Figure 1-13** Apple IIc Plus external features, front



■ **Figure 1-14** Apple IIc Plus external features, back



The keyboard

The Apple IIc Plus computer's primary input device is the **keyboard**, shown in Figure 1-15. The keyboard has a 62-key typewriter layout with both uppercase and lowercase characters, and can generate all 128 standard ASCII characters. A reset key, keyboard switch, volume control, disk-use light, and power light are also located on the front of the computer.

■ **Figure 1-15** Front of the Apple IIc Plus with Apple Standard Keyboard layout

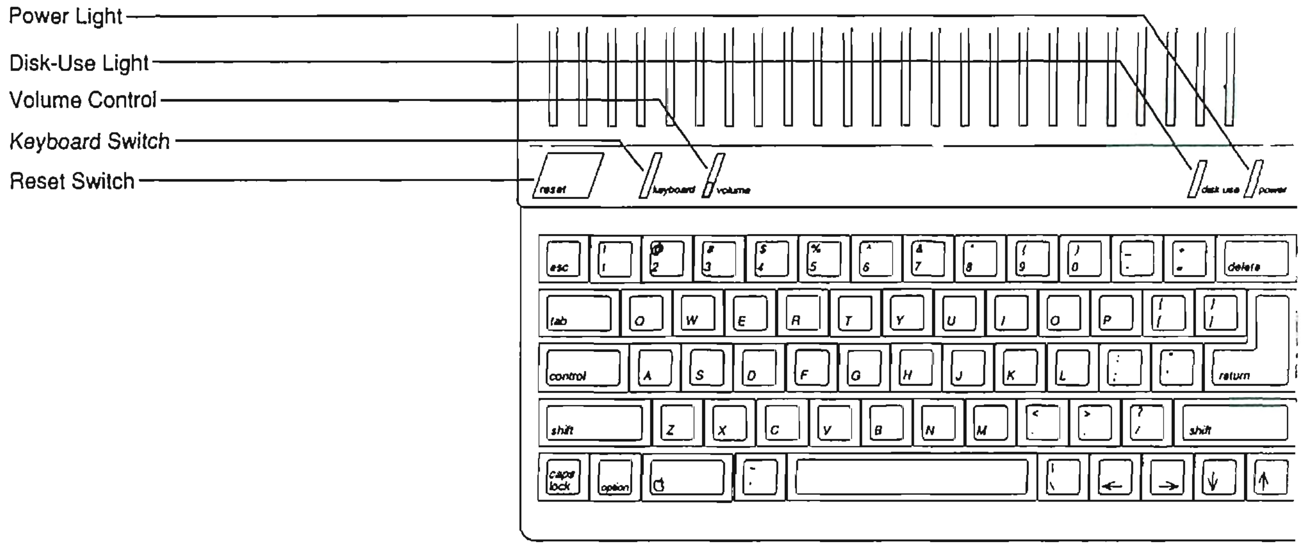


Table 1-3 lists the characteristics of all Apple IIc Plus keyboards and front panels.

Operating features

The Apple IIc Plus keyboard has automatic repeat on all character keys. This means that if you hold the key down longer than about a second, the character it generates repeats until you let up the key. It also has two-key rollover, which means if you press a key before releasing the key you pressed before it, the second character enters the computer as though you had released the previous key.

■ Table 1-3 Apple IIc Plus keyboard specifications

Number of keys	62
Character encoding	ASCII
Number of codes	128
Features	Automatic repeat, two-key rollover
Cursor movement keys	Left Arrow, Right Arrow, Down Arrow, Up Arrow, Return, Delete, Tab
Modifier keys	Control, Shift, Caps Lock
Special function keys	Command, Option, Esc (Escape)
Front-panel switches	Reset, volume control, keyboard switch
Front-panel lights	Power light, disk-use light

Special function keys

The Apple IIc Plus keyboard has three special function keys: Command, Option, and Esc. These keys have the same functions as the same keys on the original Apple IIc. The Command key is marked with the outline of an apple. On the original Apple IIc, the Option key is marked with a filled-in apple; on the Apple IIc Plus, this key is marked with the word *option*.

Front panel switches

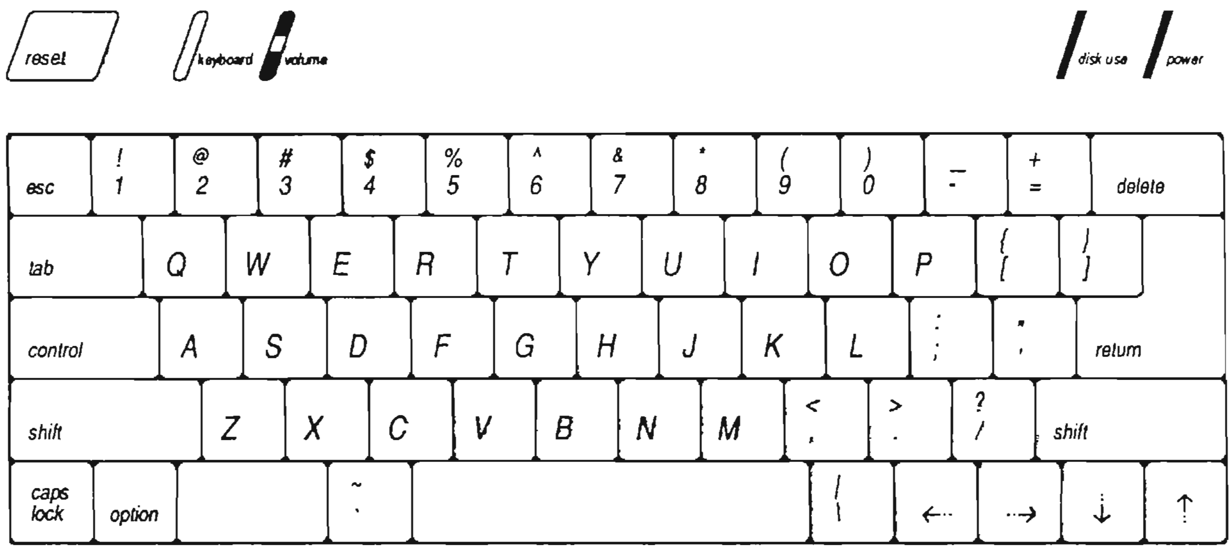
Above the keyboard of the Apple IIc Plus are the Reset switch, the volume control, and the keyboard layout switch.

The Reset switch and keyboard layout switch operate in exactly the same way as these switches did on the original Apple IIc. The keyboard layout switch is located in the position occupied by the 80/40-column switch on the original Apple IIc. The Apple IIc Plus has no 80/40-column switch; this function is controlled by software on the Apple IIc Plus computer.

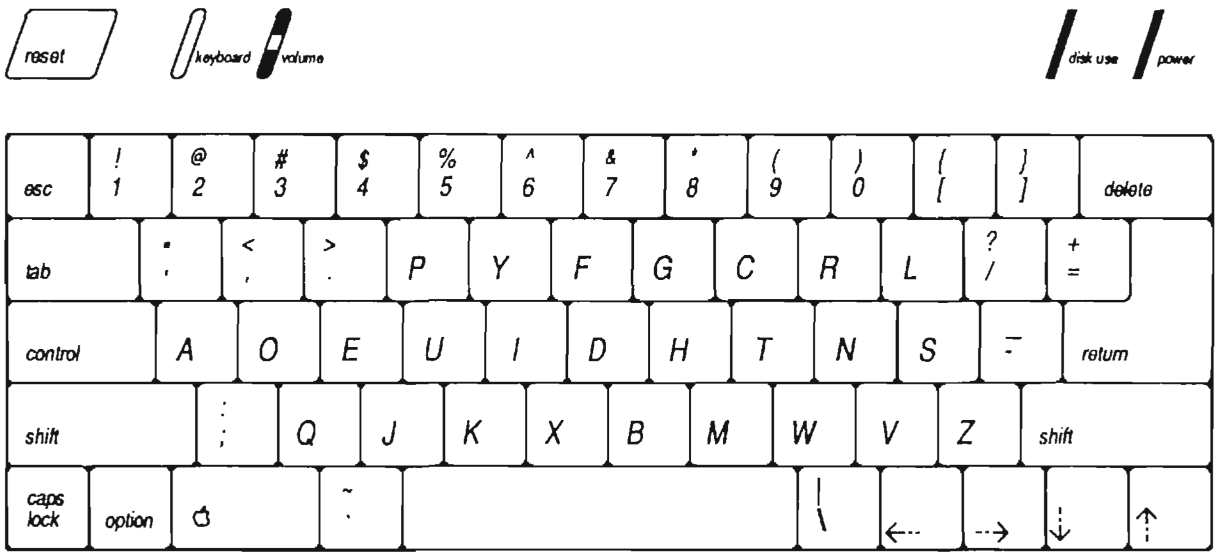
The volume control is located in the position occupied by the keyboard layout switch on the original Apple IIc. This sliding control replaces the volume control knob on the side of the original Apple IIc.

The Apple IIc Plus comes from the factory with the labels on the keycaps arranged in the Sholes positions. If you normally use the Dvorak keyboard layout, and wish to change the key caps, *gently* pry up the keycaps from the keyboard and rearrange and replace them in their Dvorak positions.

■ **Figure 1-16** Key assignments for the Apple IIc Plus keyboard, with keyboard switch up (standard Sholes position)



■ **Figure 1-17** Key assignments for the Apple IIc Plus keyboard, with keyboard switch down (Dvorak position)



Disk-use and power lights

The disk-use and power lights work exactly the same on the Apple IIc Plus as on earlier versions of the Apple IIc.

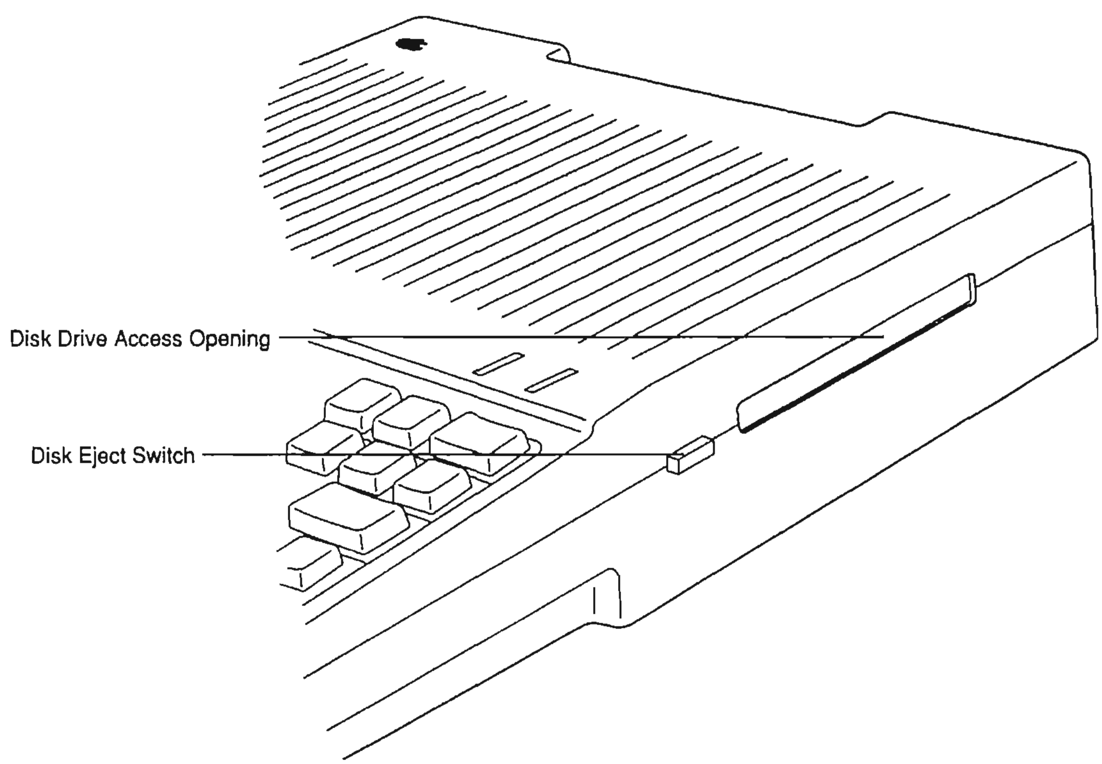
The speaker

The speaker in the Apple IIc Plus works the same way that the speaker in earlier versions does. However, the audio output jack is gone and the volume control is now positioned on the front of the computer, above the keyboard.

The disk-access opening for the internal disk drive

The disk-access opening for the internal disk drive is located in the same place as on earlier versions of the Apple IIc. A disk eject button is mounted on the left side of the opening; this button works only while the Apple IIc Plus is switched on. In addition, software can eject disks from the internal 3.5-inch disk drive by issuing a Control call to the SmartPort firmware.

■ **Figure 1-18** Apple IIc Plus internal disk drive door



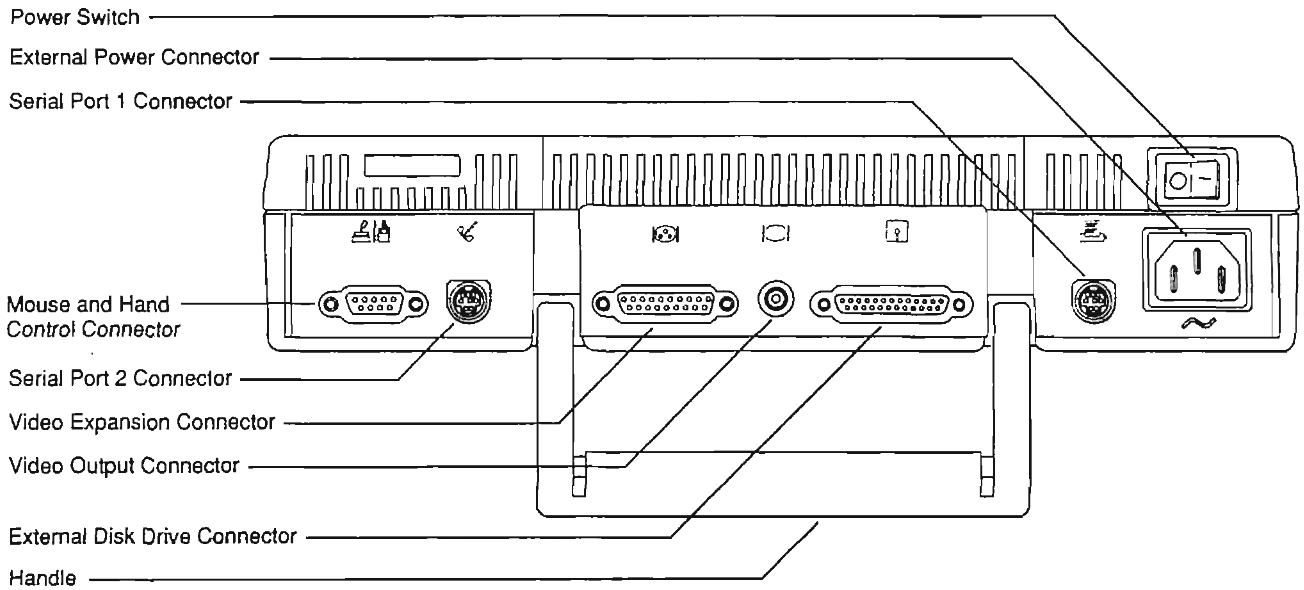
The back panel

The back panel of the Apple IIc Plus (Figure 1-19) has seven connectors and a main power switch. From left to right they are

- a 9-pin D-type (DB-9) miniature connector for connecting hand controllers, a mouse, a joystick, or some similar device
- a circular 8-pin mini-DIN connector for a modem or other serial device I/O
- a 15-pin D-type (DB-15) connector for video expansion
- an RCA-type jack for a video monitor
- a 19-pin D-type (DB-19) connector for connecting one or more external peripheral devices, such as disk drives
- a second circular 8-pin mini-DIN connector for a printer or other serial device I/O
- a standard 3-pin connector for a power cord

As for the original Apple IIc, you should move the handle of the Apple IIc Plus until it clicks into position for propping up the computer before you attach cables to the back panel connectors. Keep the handle in this position whenever the computer is running so that the computer can maintain adequate cooling airflow.

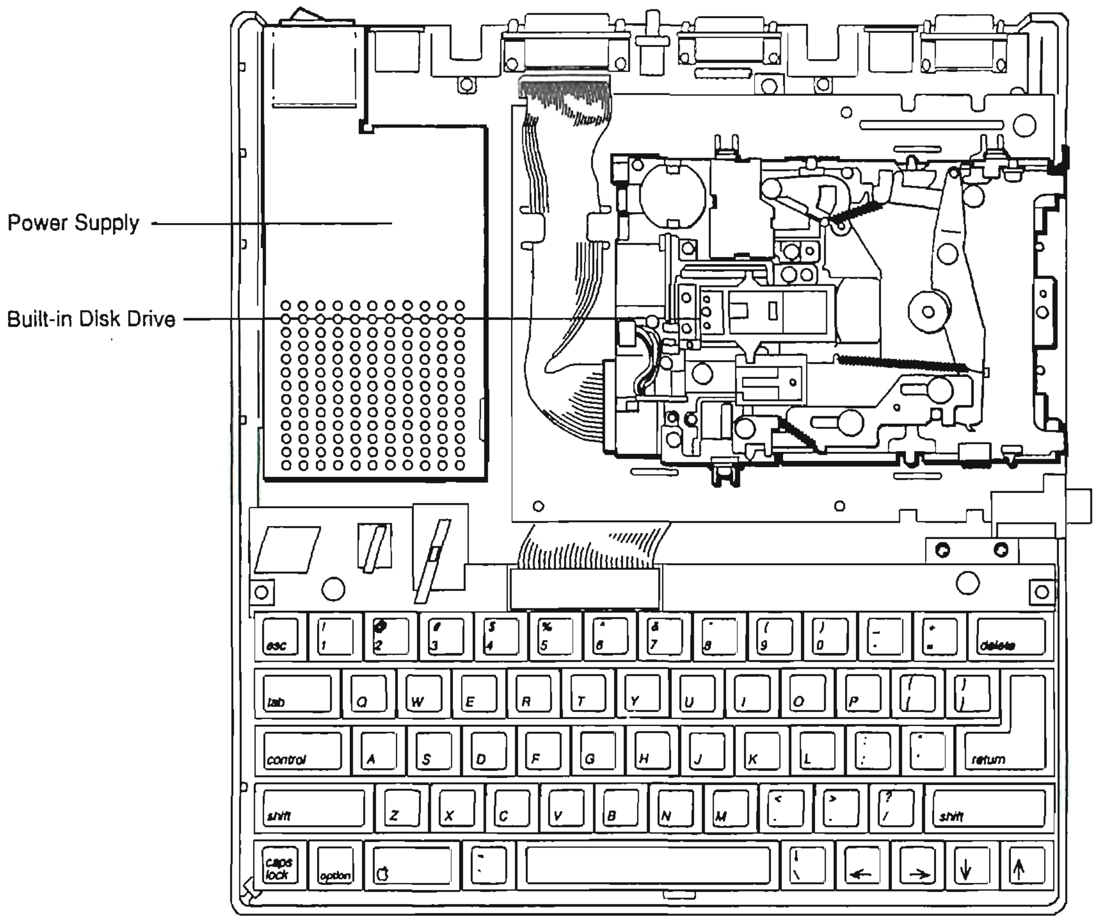
■ **Figure 1-19** Apple IIc Plus back panel connectors



The inside of the Apple IIc Plus

Figure 1-20 shows the main components inside the Apple IIc Plus computer.

■ **Figure 1-20** Inside the Apple IIc Plus



The internal power supply

The internal power supply operates from a 115 VAC input source. It provides the Apple IIc Plus with the standard Apple IIc DC voltages for the main logic board, the internal disk drive, up to three external disk drives, and the I/O signals available at the back panel. The internal power supply is shown in Figure 1-20.

The standard DC voltages are ±12 volts and +5 volts. The –5 volt supply is generated on the main logic board.

The main logic board

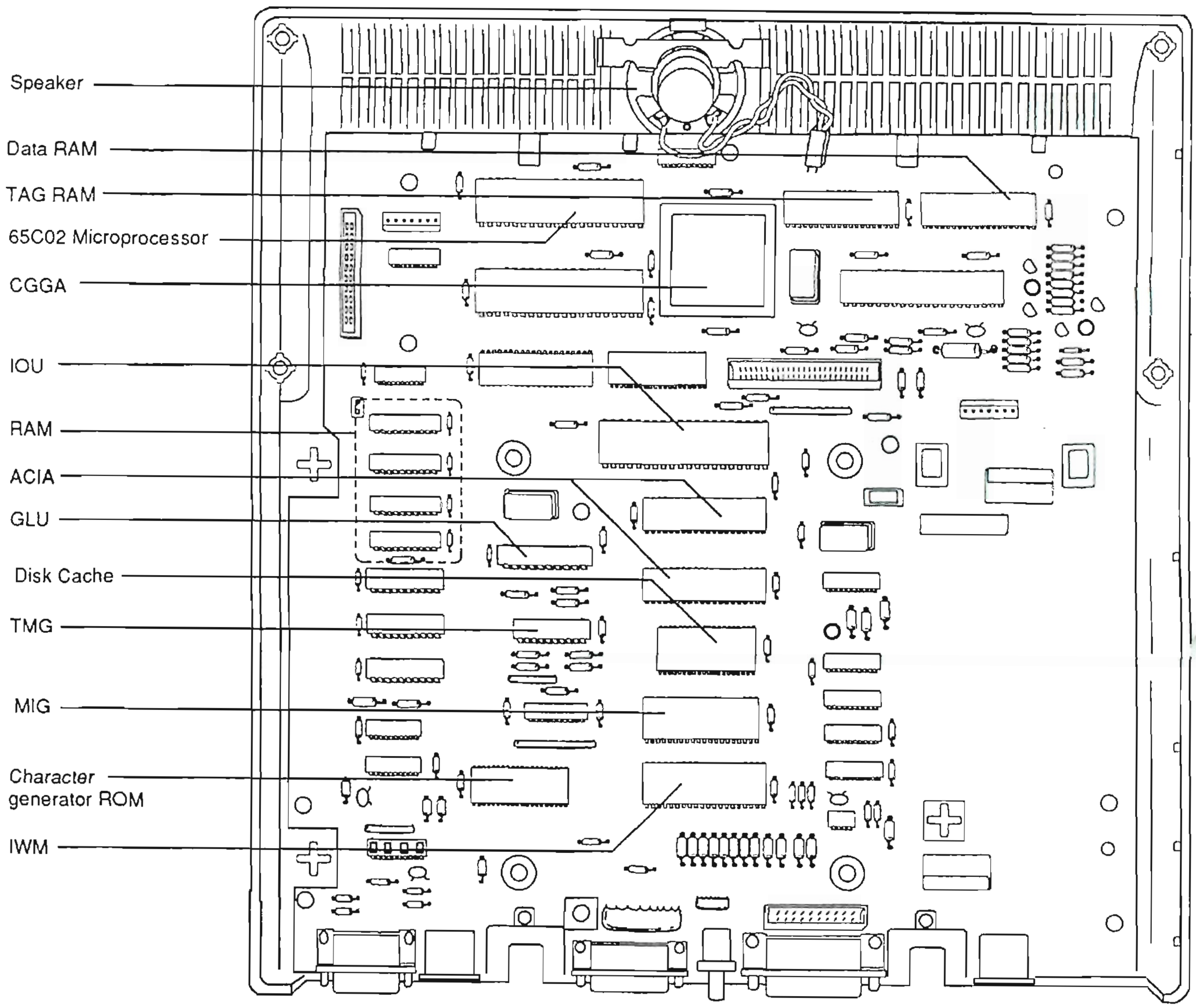
The Apple IIc Plus main logic board is mounted on the bottom of the case, as for all of the earlier versions of the Apple IIc.

Figure 1-21 shows the main logic board and the *new* integrated circuits in the Apple IIc Plus. The new circuits are

- the 4 MHz 65C02 microprocessor
- the 16 MHz crystal oscillator
- the 84-pin accelerator IC, the cache glue gate array (CGGA)
- the two 8 KB static RAMs for the accelerator circuit
- the custom IC for the 3.5-inch disk drive logic circuit called the *Multidrive Interface Glue* (MIG)
- the 2 KB static RAM for the 3.5-inch disk drive logic circuit

The Apple IIc Plus main logic board also provides a connector for internal modems, and a faster microprocessor than that used in previous Apple IIc computers. In the Apple IIc Plus, the 65C02 runs at either 1 MHz or 4 MHz, and performs up to 2,000,000 8-bit operations per second. These components are described in Chapter 11.

■ **Figure 1-21** Apple IIc Plus main logic board



The other circuit boards

The Apple IIc Plus contains other circuit boards that serve special purposes: a motor-speed control and read/write logic board for the disk drive, and a matrix board for detecting the position of keys pressed. This manual does not discuss these circuit boards.

Using the accelerator feature

The Apple IIc Plus accelerator, consisting of the 4 MHz 65C02, the CGGA, and a RAM cache, allows the Apple IIc Plus to operate at a much faster speed than other Apple IIc computers. This increase in speed is achieved whenever the Apple IIc Plus is processing data or code that is in the RAM cache. Firmware corresponding to ports 1, 2, 5, and 6 always runs at 1 MHz, because the code for the serial ports and disk drives must run at this speed to operate correctly. Firmware for ports 3, 4, and 7, on the other hand, can be cached and so potentially can run at 4 MHz. The user can disable the accelerator by holding down the Esc key while resetting the computer.

When writing software for the Apple IIc Plus computer, you should keep in mind that the user can elect to run the program at either the normal speed of other Apple IIc computers, or at the higher speed provided by the Apple IIc Plus. Some programs might not work satisfactorily at the faster speed of the Apple IIc Plus computer; for example, a game that reads a game paddle might not provide sufficient time for the user to act when running at the Apple IIc Plus's fast speed. You can avoid this problem by writing programs that force the processor to run at 1 MHz during critical routines such as the one that reads the game paddle. See the sections "Enabling and Disabling the Accelerator" and "The ROM Wait Routine," in Chapter 11, for more information on controlling the speed at which the Apple IIc Plus processes code.

The 65C02 microprocessor

The 65C02 is a general-purpose 8-bit CMOS microprocessor similar in operation to the 6502 used in other members of the Apple II–family of computers.

Figure 1-22 is a model of the 65C02 microprocessor's register organization. The 65C02 has one 16-bit register and five 8-bit registers.

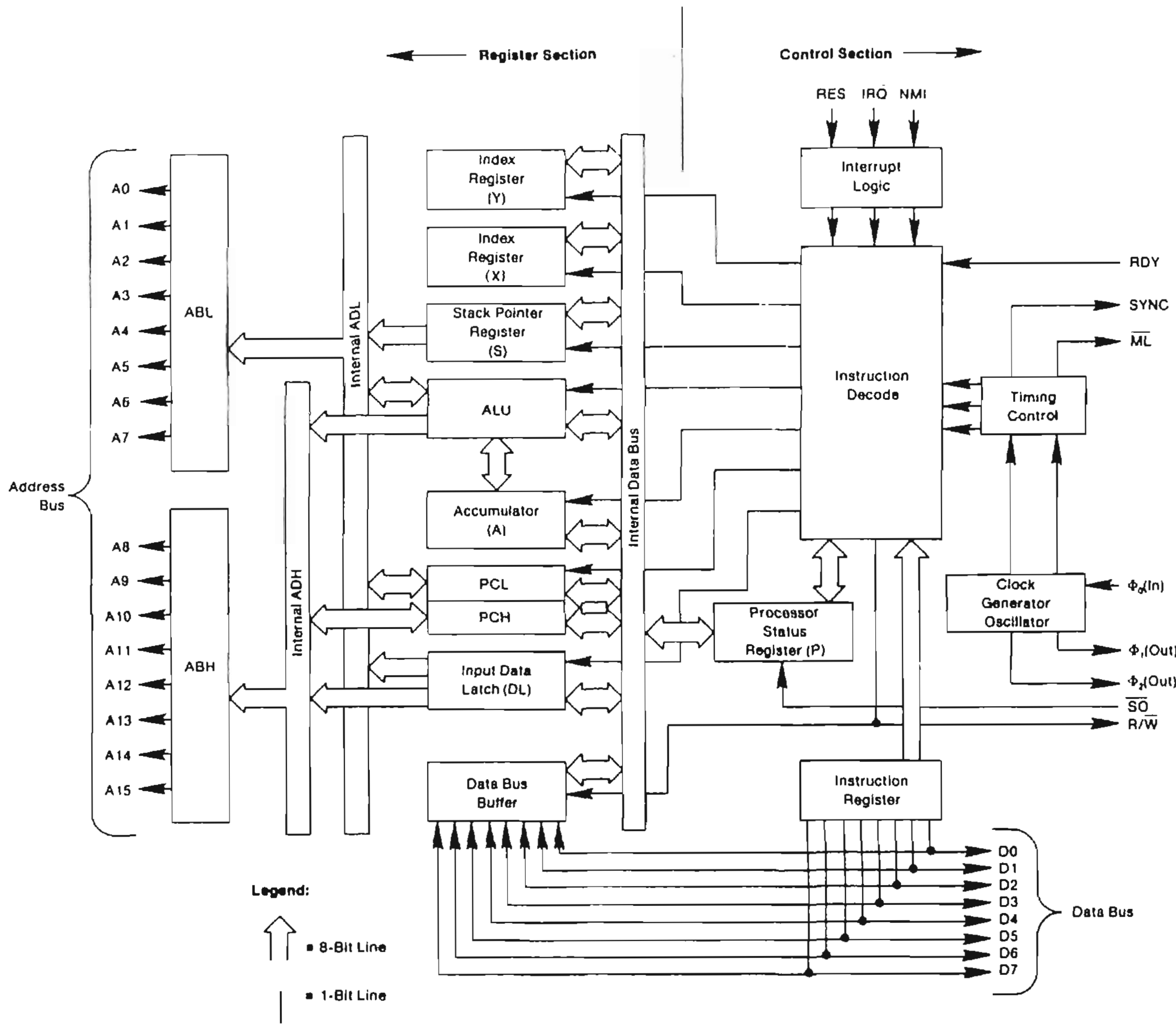
The 16-bit register is called the **program counter** (PC). It specifies the address in memory that contains the instruction the processor is currently carrying out. A 16-bit register can specify any one of 65,536 memory addresses, and so the 65C02 is said to have an address space of 65,536 locations.

The five 8-bit registers in the 65C02 are the following:

- The **A register.** The A register is like a desk top where the processor performs mathematical and logical operations on information. The A register is also called the *accumulator.*

- The **index registers,** X and Y. The processor uses these registers to modify the address where information is to be found or placed, and to pass information from one program to another.
- A **stack pointer,** or S register. The stack is a portion of memory located in page $01 that the processor uses to temporarily store data. Several 65C02 instructions either push (store) the contents of a register onto the stack, or pull (retrieve) a byte from the stack and place it in a register. The S register keeps track of the next location in memory that will be used when one of these instructions is executed.
- A **Processor Status register,** or P register. Seven of the eight bits of this register are used as flags to record the outcome of processor activities, and can be checked by later instructions to determine what has happened and what the processor should do next.

■ **Figure 1-22** Internal model of the 65C02 microprocessor (© 1982 by NCR Corporation; used by permission)



Chapter 2 **Address Map and Memory**

This chapter describes the address space in the Apple IIc and Apple IIc Plus computers, and explains how to switch between different parts of memory. ■

Overview of the address space

The 65C02 processor's 16-bit address bus provides 2^{16} unique addresses. If all of these addresses were dedicated to memory, the processor could directly address 65,536 bytes (64 KB) of RAM and ROM at one time. However, the addressing scheme used by the Apple IIc family is somewhat more complex than a one-to-one correspondence between addresses and memory locations.

For one thing, each Apple IIc family member has 128 KB of RAM plus 32 KB of ROM (16 KB in the original Apple IIc) built in. In addition, the Apple IIc and Apple IIc Plus computers use a portion of the address space to select devices and switch between portions of memory. Two of the devices selectable through address ranges are RAM and ROM; others are the asynchronous communication interface adapters (ACIAs), the IWM, and various other ICs. Because this device-select scheme uses the memory-address lines, it is referred to as *memory-mapped device selection*.

The hardware addresses that you can use are listed in Appendix B.

△ Apple IIc Plus Memory expansion: A memory expansion card that connects to the internal connector in the memory expansion Apple IIc or the Apple IIc Plus can contain up to 1 MB of RAM. Such a memory expansion card is addressed as a block device, however, not as memory. Block device I/O is discussed in Chapter 8.

The memory expansion card manufactured by Apple Computer, Inc., for use in the memory expansion version of the Apple IIc computer, *cannot* be used in the Apple IIc Plus computer. For more information on the Apple memory expansion card, see the *Apple IIc Memory Expansion Card Technical Reference*. △

In some cases, different functions have the same address—but not at the same time. The Apple IIc and Apple IIc Plus control their shared addresses by using soft switches. For example, by using soft switches to select different portions of memory, a program can take advantage of the 128 KB of built-in RAM even though the address space of the 65C02 is only 64 KB.

◆ *Note:* A **soft switch** is an address that, when accessed, controls some aspect of the computer's operation. Whenever a soft switch is used in the Apple IIc or Apple IIc Plus to control a function, its use is described in this manual under the function that the switch controls. The soft switches that control address mapping are a major topic of this chapter.

A contiguous block of 256 address locations in the 65C02 processor's address range is called a **page.** The address space of the 65C02 is divided into pages because the stack pointer and index registers are 8 bits wide, and an 8-bit register can specify only 256 different locations. Thus, page $00 consists of memory locations from 0 through 255 (hexadecimal $00 through $FF); page $01 consists of locations 256 through 511 (hexadecimal $0100 through $01FF); and so on. In this manual, all page numbers are given in hexadecimal format.

There are 256 pages of 256 bytes each in the address space of the 65C02 processor. It takes two hexadecimal digits to represent 8 bits; therefore, the first two (high-order) digits of a four-digit hexadecimal address are the page number. Do not confuse this use of the term *page* with the terms *Page 1* and *Page 2,* which are used to refer to portions of memory used to store video graphics data in the Apple IIc family.

◆ *Note:* All input and output in the Apple IIc family is address mapped—that is, specific addresses (all in the $C0 page) are allocated to each I/O device. This chapter describes only the address spaces used for I/O. For details of the built-in I/O features and firmware, refer to the descriptions in Chapters 4 through 11.

Address map

Figure 2-1 is a map of the Apple IIc family's address space; this figure shows the uses of each major block of addresses. As you can see in the figure, addresses $C000 through $C0FF are used only to access hardware, and addresses $C100 through $CFFF are used only to access ROM. All other addresses have multiple uses; each address may correspond to from two to six different locations in RAM or ROM. At any given time, each address corresponds to only one location in RAM or ROM; you use soft switches in the hardware page to control which address map the processor is currently using.

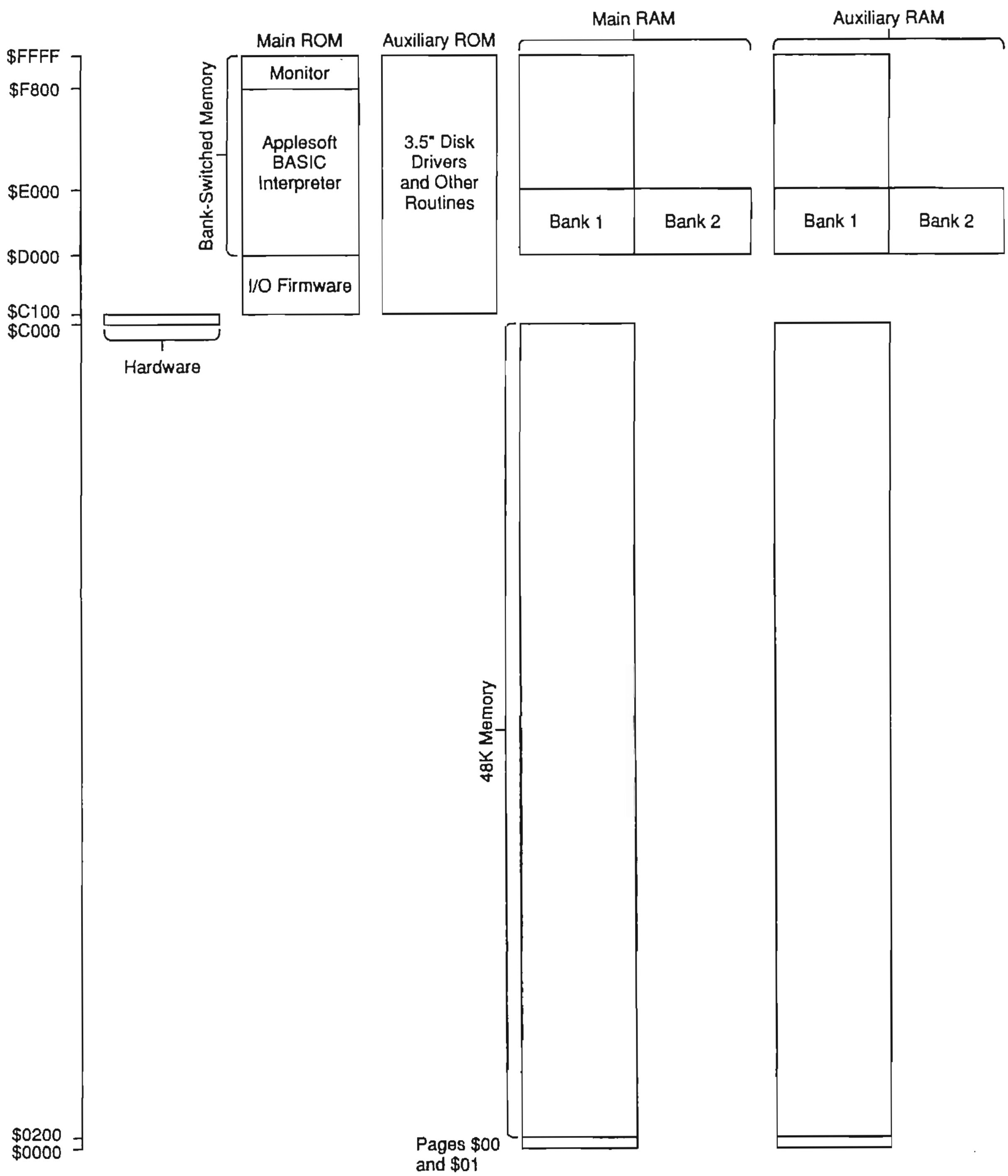
△ Original IIc — The original Apple IIc has only 16 KB of ROM; the auxiliary ROM shown in Figure 2-1 is not present in the original Apple IIc. △

◆ *Terminology:* The terminology used to describe the memory organization in Apple II–family computers can be somewhat confusing. The first 64 KB of RAM is referred to as *main RAM,* and the second 64 KB is referred to as *auxiliary RAM.* The 47.5 KB of memory space from location $0200 through location $BFFF in both main and auxiliary RAM is referred to as the *48K address space.* Together with pages $00 and $01, the 48K address space in main RAM corresponds to the entire RAM of the original Apple II and Apple II Plus computers.

The memory space from $D000 through $FFFF corresponds to the additional RAM added to the Apple II or Apple II Plus computers by a circuit card called the **language card,** and is sometimes referred to as the *language card* or *LC* in the Apple IIc family as well. The language card RAM comprises 16 KB of memory; however only 12 KB of address space is available for it (address locations $C000 through $CFFF are reserved for hardware addresses and ROM). To provide a full 16 KB of address space for the language card RAM, the address space from $D000 through $DFFF can be switched between two different 4 KB portions of RAM in both main RAM and auxiliary RAM. These portions of RAM are referred to as **Bank 1** and **Bank 2,** or sometimes as *Language Card Bank 1* and *Language Card Bank 2.*

The entire language card address space is referred to in this book as **bank-switched memory;** however, some people refer to only the $D0 page as bank-switched memory. To add to the confusion, the main RAM, auxiliary RAM, and ROM are sometimes referred to as memory banks. In this book, only Bank 1 and Bank 2 are referred to as memory banks.

■ **Figure 2-1** Apple IIc family address map



Main RAM addresses ($0000–$BFFF and $D000–$FFFF)

The area labeled *Main RAM* in Figure 2-1 is the first 64 KB of RAM; this part of RAM corresponds to the RAM in an Apple IIe that does not have an Apple IIe Extended 80-Column Text Card installed.

Auxiliary RAM addresses ($0000–$BFFF and $D000–$FFFF)

The area labeled *Auxiliary RAM* in Figure 2-1 is the second 64 KB of RAM; this part of RAM corresponds to the RAM added to an Apple IIe by the installation of an Apple IIe Extended 80-Column Text Card.

The same set of addresses is used to access main RAM and auxiliary RAM; you use the memory selector soft switches to determine which portion of RAM is actually accessed.

ROM addresses ($C100–$FFFF)

ROM contains the firmware for an Apple IIc–family computer. Addresses $C100 through $CFFF are used to access main or auxiliary ROM, depending on how the ROM selector soft switch is set. Addresses $D000 through $FFFF can access main ROM, auxiliary ROM, main RAM, or auxiliary RAM, depending on how the memory selector soft switches are set.

Pages $C1 through $CF (addresses $C100 through $CFFF) of main ROM contain I/O firmware. The Apple IIc–family I/O firmware entry points are allocated to ROM pages as follows:

- Serial port 1 (RS-232 device) firmware entry points are in page $C1.
- Serial port 2 (communication device) firmware entry points are in page $C2.
- Video output firmware entry points are in page $C3.
- Mouse firmware entry points are in page $C4 of the original Apple IIc and the Unidisk 3.5 Apple IIc.

- Memory expansion card entry points are in page $C4 of the memory expansion Apple IIc and the Apple IIc Plus.
- UniDisk 3.5 disk driver entry points are in page $C5 of the UniDisk 3.5 version and later versions of the Apple IIc. In the Apple IIc Plus, the entry points for the Apple 3.5 disk driver are located in this page as well.
- Disk II (5.25-inch disk drive) entry points are in page $C6.
- Mouse firmware entry points are in page $C7 in the memory expansion Apple IIc and in the Apple IIc Plus.

◆ *Note:* The assignment of each I/O port's entry points to a particular page of memory gives each port the same entry points as it would have if the firmware were on a circuit card in a slot. In fact, programmers sometimes refer to the Apple IIc–family I/O entry points as "slots," as in "serial port 1 is in slot 1 in the Apple IIc."

The address range of pages $D0 through $FF in main ROM contains the Applesoft BASIC interpreter and the Monitor firmware, allocated as follows:

- Pages $D0 through $F7 (addresses $D000 through $F7FF) contain the Applesoft interpreter firmware.
- Pages $F8 through $FF (addresses $F800 through $FFFF) contain the Monitor, described in Chapter 10. You can use some of the built-in Monitor routines to make input and output procedures in your assembly-language programs easier to write.

The auxiliary ROM contains a variety of routines, including some routines for the mouse, the serial ports, the UniDisk 3.5 disk drive, and the Monitor. The auxiliary ROM also contains the SmartPort routines. In addition, in the Apple IIc Plus computer, the auxiliary ROM contains all of the disk driver routines for the internal 3.5-inch disk drive and external Apple 3.5 disk drives.

▲ **Warning** There are no routines in auxiliary ROM that you can use, and there is no way for your program to determine whether main or auxiliary ROM is switched in. Whereas the firmware in the UniDisk 3.5 and memory expansion Apple IIc computers would automatically return to the main ROM if you inadvertently toggled the ROM selector switch, in the Apple IIc Plus, all the space in auxiliary ROM is used and your program will crash if you make a call to auxiliary ROM. ▲

Hardware addresses ($C000–$C0FF)

The soft switches that the Apple IIc or Apple IIc Plus and your programs use to control the computer's built-in input and output functions are all found in the $C0 address page (addresses $C000 through $C0FF). In the same range of memory are the switches for selecting blocks of memory throughout the address space. This chapter describes the address-space (memory) switches. Tables B-5 through B-9 in Appendix B list the hardware page addresses.

The hardware functions of the switches in this page fall into five basic categories:

- *Data inputs.* The only data input is location $C000, where the low-order 7 bits (bits 6 through 0) represent the keyboard key just pressed. (These data are guaranteed valid only when bit 7 is 1.)
- *Flag inputs.* Most built-in input locations are single-bit flags in the high-order (bit 7) position of their respective memory addresses. Flags have only two values: *on* (greater than or equal to 128 or $80) or *off* (less than 128 or $80).

  The mouse button inputs and the keyboard any-key-down bit are examples of flag inputs. The locations for reading soft-switch states are also of this type.
- *Strobe outputs.* The clear keyboard strobe (Chapter 5) and paddle timer strobe (Chapter 1) outputs are controlled by memory locations. If your program reads the contents of one of these locations, then the function associated with that location is activated.
- *Toggle switches.* The Apple IIc family has two toggle switches: the speaker switch and the main/auxiliary ROM selector switch. A toggle switch has only one address assigned to it; each time you access it, it changes to its other state (on or off).

  The processor cannot read the status of these toggle switches.

  Reading the speaker switch activates the toggle once. However, if you *write* to the speaker switch location, the microprocessor activates the address bus twice during successive clock cycles, causing the toggle to end up in its original state. Therefore, you should read, rather than write, to use this switch.
- *Soft switches.* Soft switches are two-position switches turned on by accessing one address and turned off by accessing another address. Most of these switches have a third address associated with them for reading the state of the switch.

  There are eight soft switches that select different combinations of bank-switched memory. Four of these eight switches require that your program read them twice in succession to activate them.

Bank-switched memory

The memory areas described in this section are called *bank-switched memory* (see Figure 2-1) because so many distinct portions (banks) of memory—two portions of ROM and up to four portions of RAM—are accessed with the same group of addresses. Pages $00 and $01, at the low end of memory, are included here because the two sets of them—one in main RAM and one in auxiliary RAM—are controlled by the same switches as the high-address bank-switched memory. Page $00 and $01 are switched this way so that system software running in the bank-switched memory space can maintain its own zero page and stack independent of the 48K memory space.

The soft switches used for selecting between main and auxiliary 48K memory space are discussed in the section "48K Memory," later in this chapter.

Page allocations

Pages $00 and $01 are used by many of the 65C02 instructions. The ROM and RAM addresses in bank-switched memory are usually occupied by system software such as interpreters, compilers, and operating systems.

Page $00 (one-byte addresses)

Several of the 65C02 microprocessor's addressing modes—for example, indirect addressing—require the use of addresses in page $00, or **zero page.** However, the Monitor, the interpreters, and the operating systems all make extensive use of page $00, too. One way to avoid conflicts is to use only those page-$00 locations not already used by these other programs. But there is another way.

As you can see from Table B-1 in Appendix B, page $00 is pretty well used up, except for a few bytes here and there. Rather than trying to squeeze your data into an unused corner, you may prefer a safer alternative: turn off interrupts, save the contents of part of page $00, use that part, then restore the previous contents to page $00, restore interrupts to their previous state, and then pass control to another program.

Page $01 (the 65C02 stack)

The 65C02 microprocessor uses page $01 as its stack—a place where it can store subroutine return addresses, in last-in, first-out (LIFO) sequence. Programs can also use the stack for temporary storage of registers (via push and pull instructions). However, programs should use the stack carefully.

Pages $D0–$FF (ROM and RAM)

The memory address space from locations $D000 through $FFFF is used for both ROM and RAM. The 12 KB of main ROM that can be accessed using these addresses contains the Monitor and the Applesoft BASIC interpreter. The auxiliary ROM in this address space contains SmartPort firmware, the disk driver for the internal 3.5-inch disk drive (Apple IIc Plus only), and miscellaneous routines. The soft switches that you can use to control the area of memory accessed by these addresses are discussed in the following section, "Using Bank-Switched–Memory Selector Switches."

The RAM in pages $D0 through $FF is normally used for storing computer languages, such as Pascal, or operating systems, such as ProDOS®.

There are 16 KB of main RAM that can be accessed by addresses in the range $D000 through $FFFF. In order to accomplish this feat, the lower 4 KB of addresses (pages $D0 through $DF) serve double duty: depending on the setting of a soft switch, these addresses access one of two 4 KB blocks of main RAM. These two blocks of RAM are referred to as *main RAM Bank 1* and *main RAM Bank 2.*

There are also 16 KB of auxiliary RAM that can be accessed by addresses in the range $D000 through $FFFF. As for main RAM, the addresses in the range $D000 through $DFFF can be used to access two different 4 KB blocks of memory, referred to as *auxiliary RAM Bank 1* and *auxiliary RAM Bank 2.*

Using bank-switched–memory selector switches

You select banks of memory in the same way you control other functions in the Apple IIc family: by using soft switches. The bank-switched–memory selector switches do four things:

- select whether the address range for bank-switched memory accesses RAM or ROM, or accesses ROM for reads and RAM for writes

- when RAM is selected for reads, either allow or inhibit (write-protect) writing to the RAM
- when RAM is selected, select either main RAM or auxiliary RAM
- select the first or second 4 KB bank of RAM in the address space $D000 through $DFFF

◆ *Note:* There is also a selector switch to toggle between main and auxiliary ROM; however, there are no routines in auxiliary ROM that you can use, and switching to auxiliary ROM will cause your program to crash.

▲ **Warning** Do not use soft switches without careful planning. Careless switching between RAM and ROM is almost certain to have catastrophic effects on your program. ▲

Table 2-1 shows the addresses of the soft switches for selecting all allowed combinations of reading and writing in this memory space, and the addresses of the locations to read the switch settings. Figures 2-2 through 2-8 illustrate how to select the combinations and what the resulting status of each switch is.

To make sure you do not inadvertently remove write protection from bank-switched RAM, the four write-enable addresses require that you read them twice in succession (indicated by *RR* in Table 2-1).

Because the AltZP switch shares the read keyboard address, you must write (*W* in Table 2-1) to its locations to change the switch setting.

To find out which way a switch is set, read the appropriate location and then check bit 7 (this action is shown as *R7* in Table 2-1).

Note that there is no way to check whether write protection is on or off.

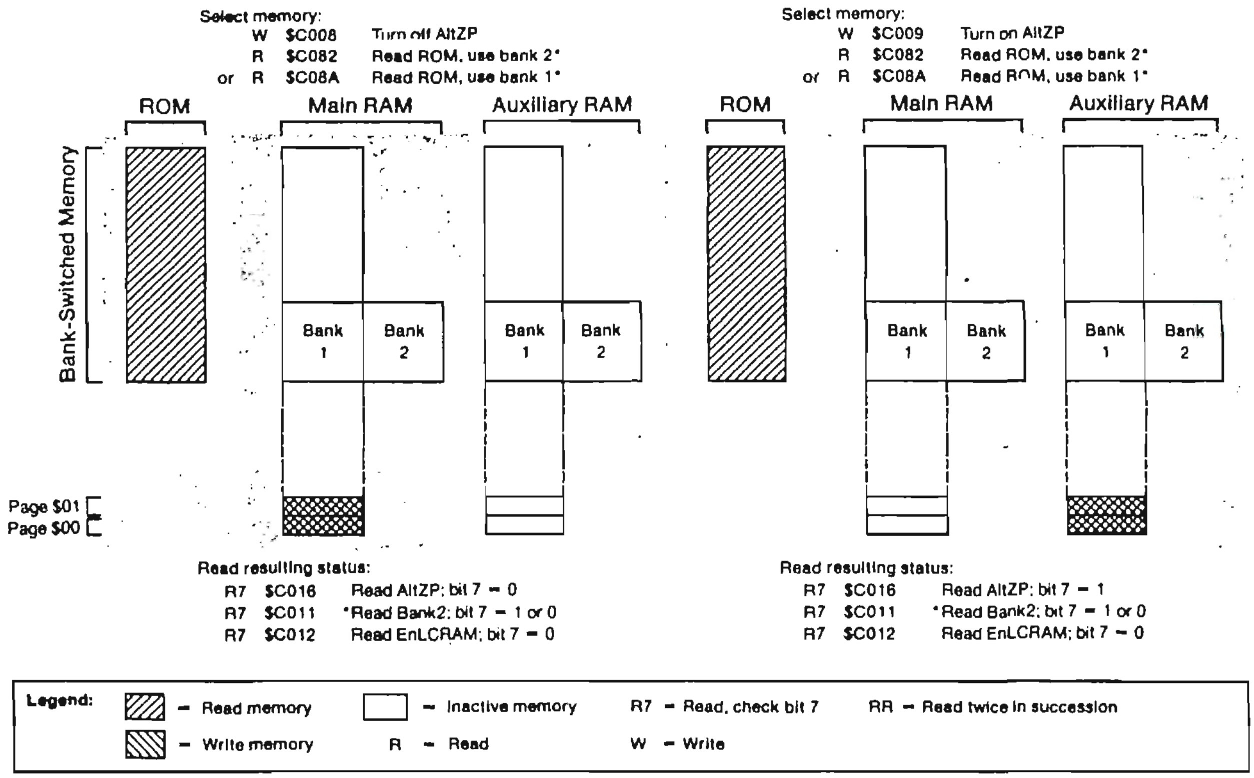
△ Important You can't read main RAM and write to auxiliary RAM (or vice versa); if you select either RAM for reading, you get that one for writing as well. However, you can read ROM and write RAM (Figures 2-3 and 2-4), which makes it easy to transfer firmware to bank-switched RAM if you want to use it with a program there. △

■ **Table 2-1** Bank-switched memory selector switches

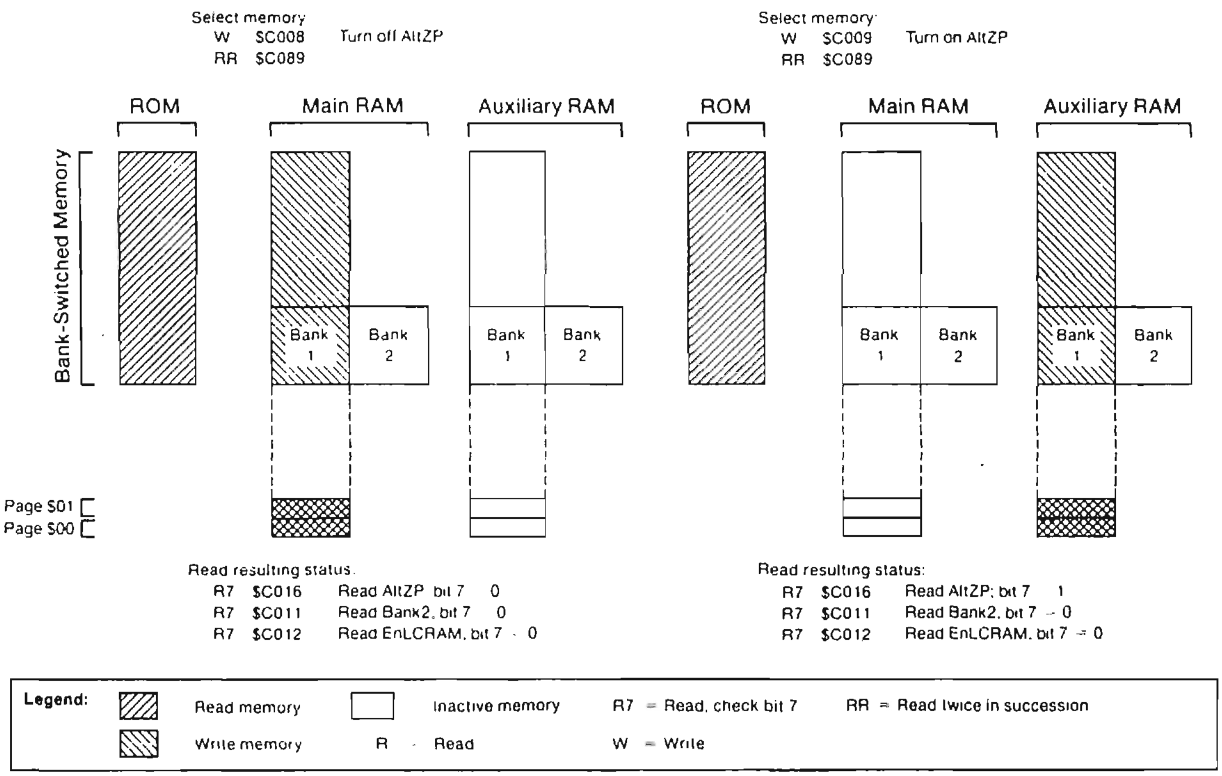
Name	Action	Hex	Dec	Function
	R/W	$C02x	49184–49199	Toggle between main and auxiliary ROM*
	R	$C080	49280	Read RAM; no write; use $D000 Bank 2
	RR	$C081	49281	Read ROM; write RAM; use $D000 Bank 2
	R	$C082	49282	Read ROM; no write; use $D000 Bank 2
	RR	$C083	49283	Read and write RAM; use $D000 Bank 2
	R	$C088	49288	Read RAM; no write; use $D000 Bank 1
	RR	$C089	49289	Read ROM; write RAM; use$D000 Bank 1
	R	$C08A	49290	Read ROM; no write; use $D000 Bank 1
	RR	$C08B	49291	Read and write RAM; use $D000 Bank 1
RdBnk2	R7	$C011	49169	Read whether $D000 Bank 2 (1) or Bank 1 (0)
RdLCRAM	R7	$C012	49170	Read RAM (1) or ROM (0)
AltZP	W	$C008	49160	Off: Use main RAM, page $00 and page $01
AltZP	W	$C009	49161	On: Use auxiliary RAM, page $00 and page $01
RdAltZP	R7	$C016	49174	Read whether auxiliary (1) or main (0) RAM

* Switching to auxiliary ROM will cause your program to crash.

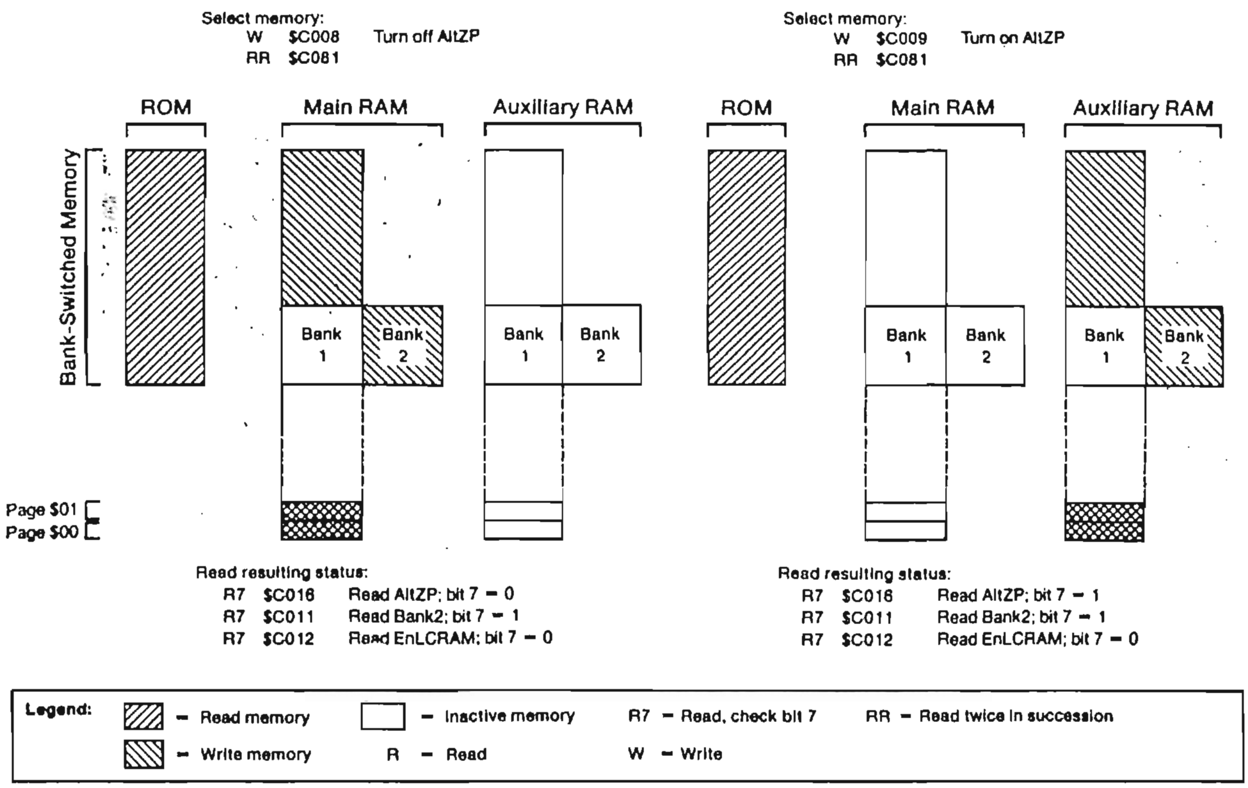
■ **Figure 2-2** Read ROM



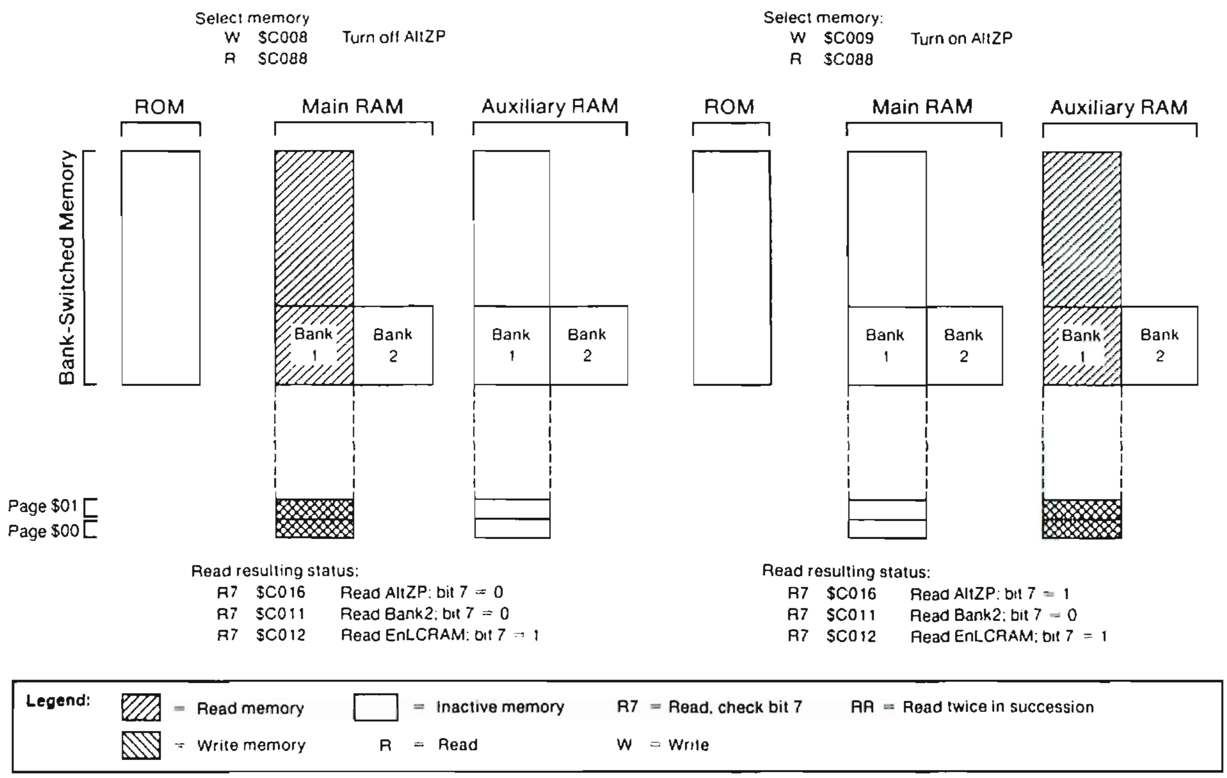
■ **Figure 2-3** Read ROM, write RAM, and use $D000 Bank 1



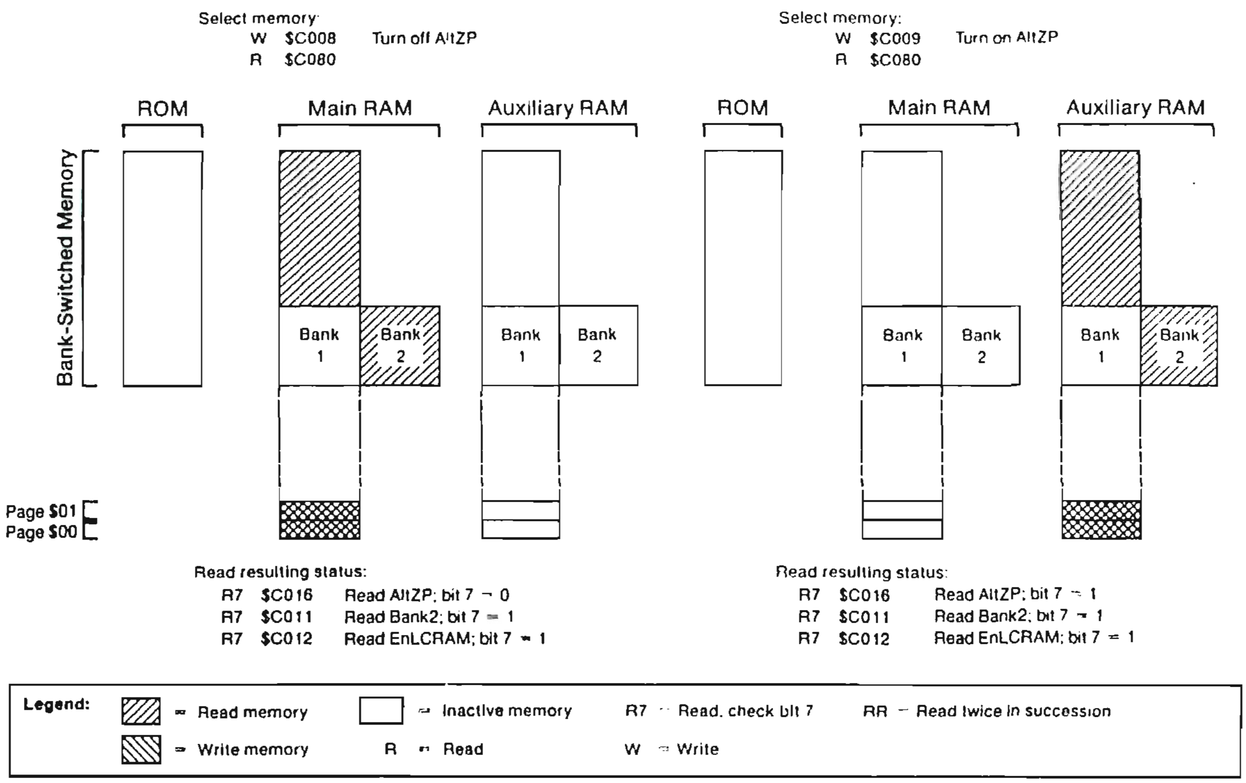
■ Figure 2-4 Read ROM, write RAM, and use $D000 Bank 2



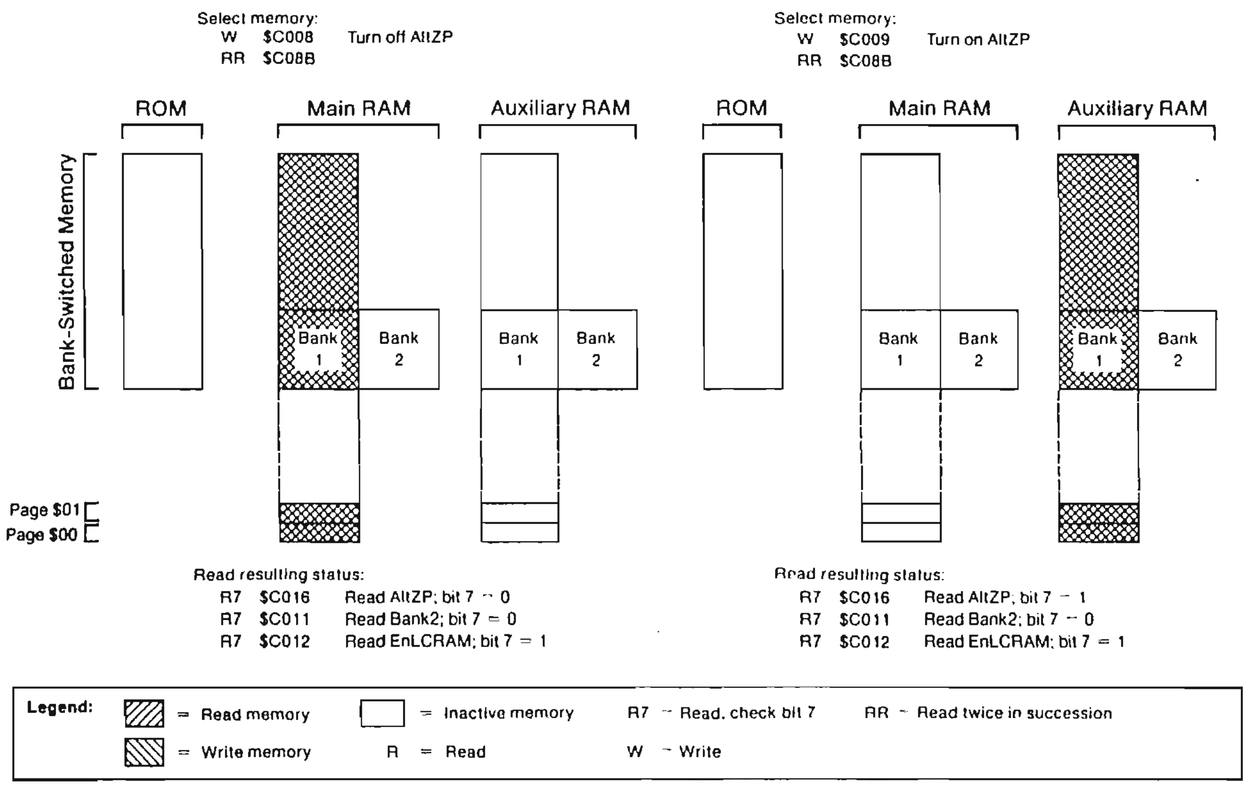
■ **Figure 2-5** Read RAM and use $D000 Bank 1



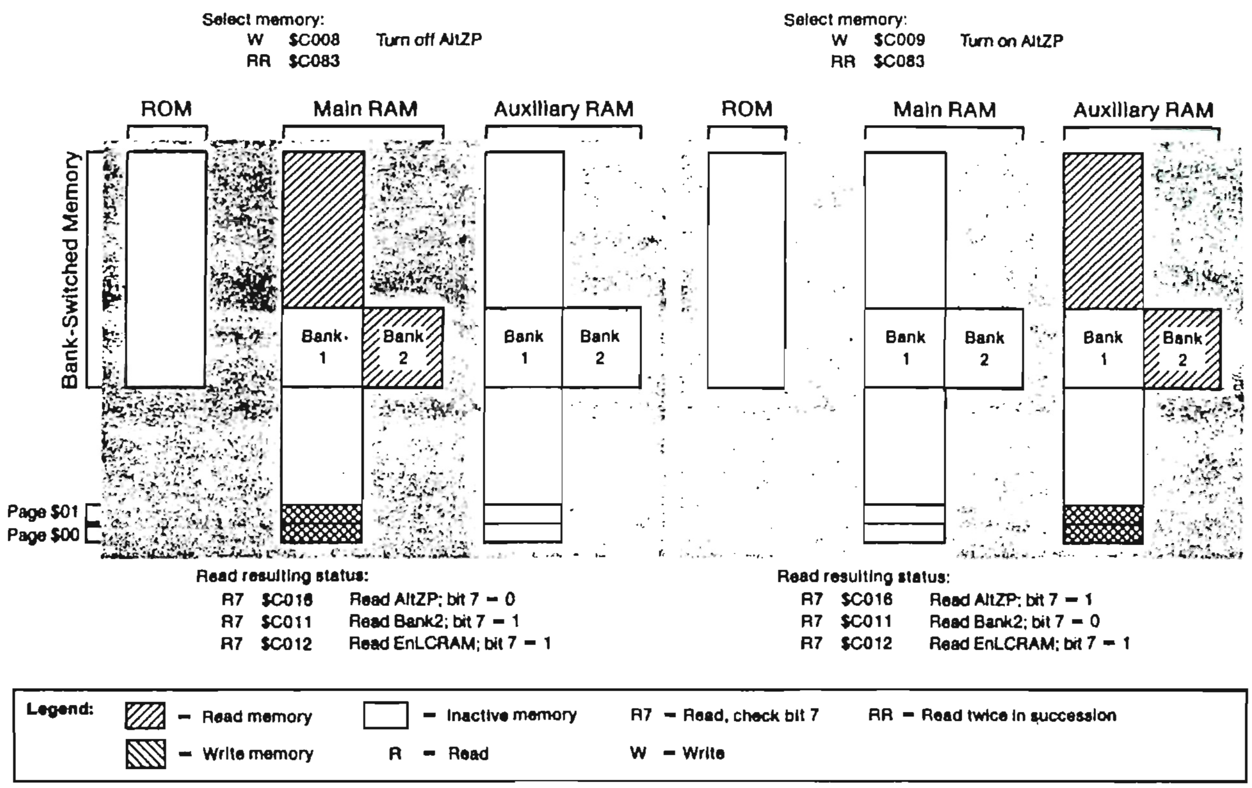
■ **Figure 2-6** Read RAM and use $D000 Bank 2



■ **Figure 2-7** Read and write RAM and use $D000 Bank 1



■ **Figure 2-8** Read and write RAM and use $D000 Bank 2



48K address space

The 48K address space (which actually occupies 47.5 KB of address space) extends from location $0200 to location $BFFF (see Figure 2-9) in both main and auxiliary RAM. The amount of storage available for your use in this address space depends on what language or operating system you are using, and on what video display needs your program has.

Page allocations

Most of the Apple IIc family's 48K address space is available for storing your programs and data. However, a few memory pages are reserved for the use of the Monitor firmware, the Applesoft BASIC interpreter, and whatever video display you may select.

△ Important — The system does not prevent your using these pages, but if you do use them, you must be careful not to disturb the system data they contain. △

Page $02 (the input buffer)

The GetLn input routine uses page $02 as its keyboard-input buffer. The size of this buffer (256 bytes) sets the maximum size of input strings read by Applesoft or the Monitor. If you know that you won't be typing any long input strings (more than, say, 30 characters), you can store temporary data at the upper end of page $02.

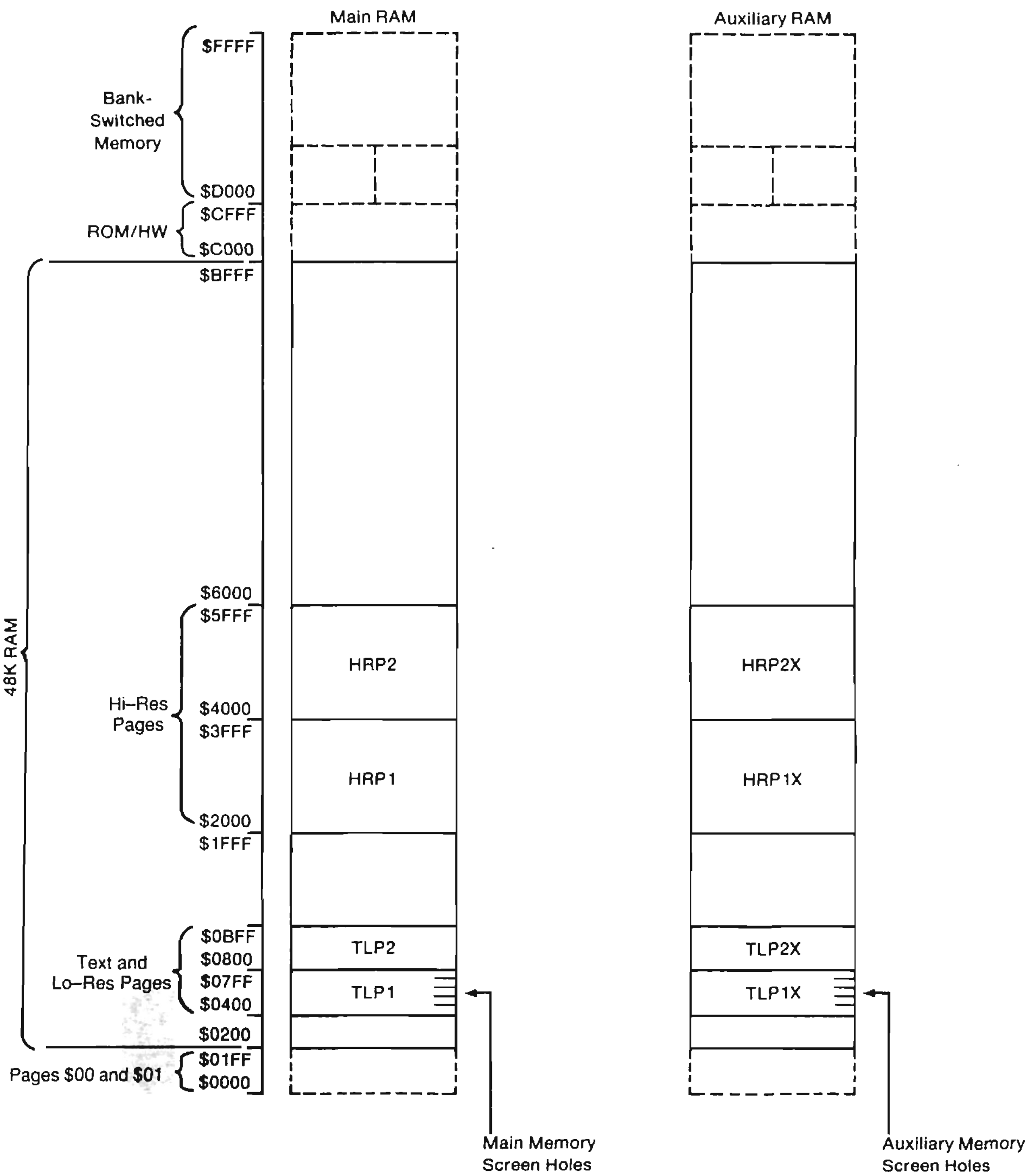
Page $03 (global storage and vectors)

The Monitor and operating systems use parts of page $03 for global storage and vectors. Table 3-1, in Chapter 3, shows the part of page $03 the built-in firmware uses. Table B-2 in Appendix B lists the complete contents of page $03.

Pages $04–$07 (text and Lo-Res graphics Page 1)

The most often used display buffer is text and Lo-Res graphics Page 1 (TLP1 in Figure 2-9), which occupies main memory pages $04 through $07. It is not usable for program and data storage if you are using Monitor routines or Applesoft, or with almost any other program that uses text or Lo-Res graphics display.

■ **Figure 2-9** 48K address map



Text and Lo-Res graphics Page 1X (TLP1X) is an identical display page occupying auxiliary memory pages $04 through $07. TLP1 and TLP1X are used together to produce 80-column text display.

There are 128 locations in pages $04 through $07 (64 in main RAM, 64 in auxiliary RAM) that are not displayed on the screen. These locations are called **screen holes.** Tables B-3 and B-4 in Appendix B list the screen holes used by the Apple IIc family.

▲ **Warning** The screen holes are reserved for use by the built-in firmware. If you use the screen holes, you might interfere with the operation of the firmware. ▲

Pages $08–$0B (text and Lo-Res graphics Page 2)

Text and Lo-Res graphics Page 2 (TLP2) occupies main memory pages $08 through $0B. Most programs do not use TLP2 for displays, but it is there for display use if required.

Text and Lo-Res graphics Page 2X (TLP2X) is an identical display buffer occupying pages $08 through $0B in auxiliary memory.

Note that Apple IIc-family firmware does not provide a way to use the second pair of text and Lo-Res graphics pages for 80-column text display.

Page $08 (communication port buffers)

Serial port 2 uses the first half of auxiliary memory page $08 (addresses $0800 through $087F) as a keyboard input buffer, and the second half of the page (addresses $0880 through $08FF) as a serial input buffer. These buffers increase the data transfer rates possible with the serial communication port. See the section "Firmware Handling of Interrupts," in Chapter 3, for a description of how to use these features. If your program does not use this page for buffers, it can use it as part of TLP2X.

Pages $20–$3F (Hi-Res graphics Page 1)

The primary Hi-Res graphics display buffer, Hi-Res graphics Page 1 (HRP1), occupies the 32 memory pages from $20 through $3F (locations $2000 through $3FFF). If your program doesn't use Hi-Res graphics, this area is usable for programs or data.

Hi-Res graphics Page 1X (HRP1X) is an identical display page occupying auxiliary memory pages $20 through $3F.

The Apple IIc-family can display Double Hi-Res graphics by interleaving HRP1 and HRP1X.

See the section "Graphics Modes," in Chapter 6, for a discussion of Hi-Res and Double Hi-Res graphics.

Pages $40–$5F (Hi-Res graphics Page 2)

Hi-Res graphics Page 2 (HRP2) occupies main memory pages $40 through $5F (locations $4000 through $5FFF). Most programs use this area for program or data storage, but it is also available as a second Hi-Res page.

Hi-Res Page 2X (HRP2X) occupies auxiliary memory pages $40 through $5F.

Apple IIc–family firmware provides Hi-Res graphics routines for HRP1 and HRP2 only.

Using 48K address space switches

Two switches select whether addresses in the 48K address space access main or auxiliary RAM: RAMRd determines which to use for reading, and RAMWrt determines which to use for writing. When these switches are on, they select auxiliary RAM. When they are off, they select main RAM. (This discussion assumes that the 80Store switch, used to control display memory, is off.) See the section "Using Display Memory Switches," later in this chapter, for more information on these switches.

Each switch has three locations assigned to it (see Table 2-2): one to turn it on, one to turn it off, and a third to read its state. Because the memory locations for turning the switches on and off are shared with keyboard reading functions, you must write to these addresses to use them for memory switching. For each switch, you can read bit 7 at the third location to check whether the switch is on or off. If the switch is on, bit 7 is 1; if the switch is off, bit 7 is 0.

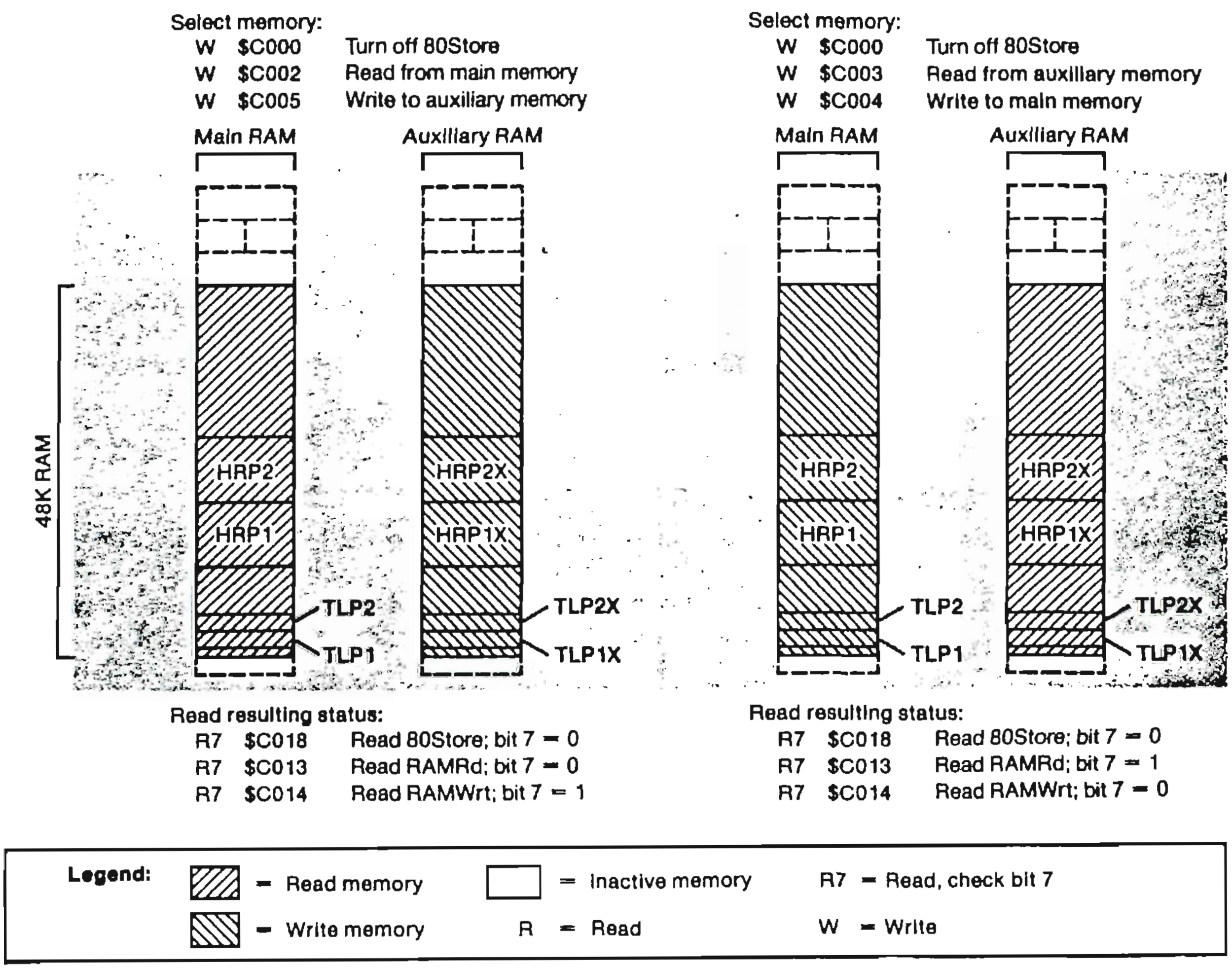
■ **Table 2-2** 48K address-space switches

Name	Action	Hex	Dec	Function
RAMRd	W	$C002	49154	Off: Read main 48K RAM
RAMRd	W	$C003	49155	On: Read auxiliary 48K RAM
RdRAMRd	R7	$C013	49171	Read whether main (0) or aux. (1)
RAMWrt	W	$C004	49156	Off: Write to main 48K RAM
RAMWrt	W	$C005	49157	On: Write to auxiliary 48K RAM
RdRAMWrt	R7	$C014	49172	Read whether main (0) or aux. (1)

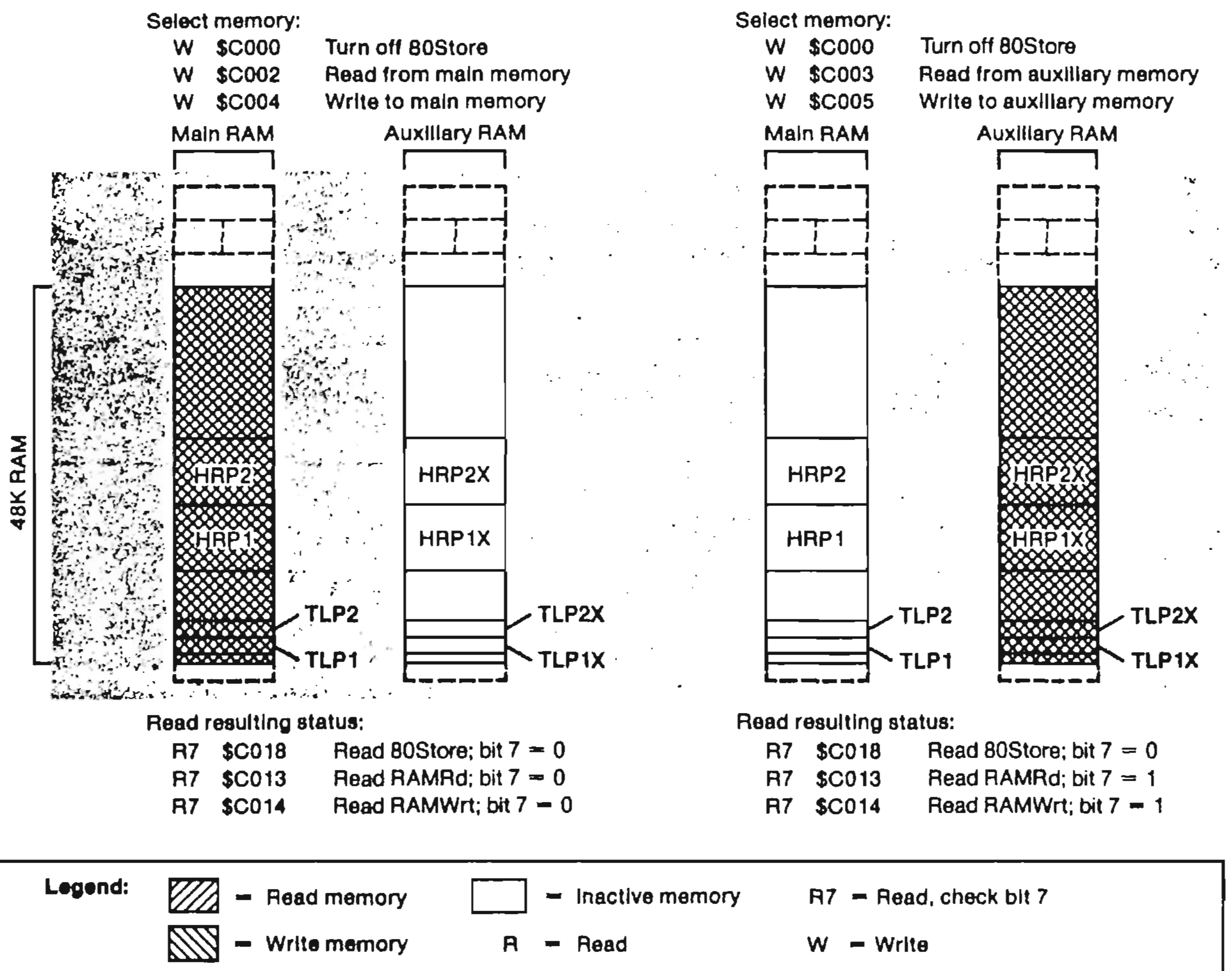
Note: 80Store must be off to switch all memory in this range, including display memory (Table 2-6).

Figures 2-10 and 2-11 illustrate how the switches work.

■ **Figure 2-10** 48K address selection, split pairs



■ **Figure 2-11** 48K address selection, one side only



Transfers between main and auxiliary memory

If you want to write assembly-language programs that use auxiliary memory but you don't want to manage the auxiliary memory yourself, you can use the built-in 48K memory transfer routines, MoveAux and Xfer. These routines (listed in Table 2-3) make it possible to move between main and auxiliary memory without having to manipulate the soft switches described earlier in this chapter.

△ Important — MoveAux and Xfer make it easier to use auxiliary memory, but they do not protect you from errors. You still have to plan your use of auxiliary memory to avoid catastrophic effects on your program. △

■ **Table 2-3** 48K address space transfer routines

Name	Action	Hex	Function
MoveAux	JSR	$C311	Move data blocks between main and auxiliary 48K address space
XFer	JMP	$C314	Transfer program control between main and auxiliary 48K address space

Transferring data

In your assembly-language programs, you can use the built-in routine MoveAux to copy blocks of data from main memory to auxiliary memory or from auxiliary memory to main memory. Before calling this routine, you must put the data addresses into byte pairs in page $00 and set or clear the carry bit to select the direction of the move.

▲ **Warning** — Don't try to use MoveAux to copy data in bank-switched memory (page $00, page $01, or pages $D0 through $FF). MoveAux uses page $00 during the copy. ▲

The pairs of bytes you use for passing addresses to this routine are called A1, A2, and A4, and they are used for parameter passing by several built-in routines. The addresses of these byte pairs are shown in Table 2-4.

Put the addresses of the first and last bytes of the block of memory you want to copy into A1 and A2. Put the starting address of the block of memory you want to copy the data to into A4.

■ **Table 2-4** Parameters for MoveAux routine

Name	Location	Parameter passed
Carry		1 = Move from main to auxiliary memory. 0 = Move from auxiliary to main memory.
A1L	$3C	Source starting address, low-order byte.
A1H	$3D	Source starting address, high-order byte.
A2L	$3E	Source ending address, low-order byte.
A2H	$3F	Source ending address, high-order byte.
A4L	$42	Destination starting address, low-order byte.
A4H	$43	Destination starting address, high-order byte.

◆ *Note:* The X, Y, and A registers are preserved.

To copy data from main memory to auxiliary memory, set the carry bit (SEC instruction); to copy data from auxiliary memory to main memory, clear the carry bit (CLC instruction).

When you make the subroutine call to MoveAux, the subroutine copies the block of data as specified by the A register and the carry bit. When it is finished, the A, X, and Y registers are just as they were when you called it.

Transferring control

You can use the built-in routine XFer to transfer control to and from program segments in auxiliary memory. You must set up three parameters before using XFer: the address of the routine you are transferring to the direction of the transfer, which zero page and stack you want to use (see Table 2-5).

■ **Table 2-5** Parameters for XFer routine

Name	Location	Parameter passed
Carry		1 = Transfer from main to auxiliary memory.
		0 = Transfer from auxiliary to main memory.
Overflow		1 = Use page $00 and stack in auxiliary memory.
		0 = Use page $00 and stack in main memory.
	$03ED	Program starting address, low-order byte.
	$03EE	Program starting address, high-order byte.

◆ *Note:* The X, Y, and A registers are preserved.

Put the transfer address into the two bytes at locations $03ED and $03EE, with the low-order byte first, as usual. The direction of the transfer is controlled by the carry bit: set the carry bit to transfer to a program in auxiliary memory; clear the carry bit to transfer to a program in main memory.

Use the overflow bit to select which page $00 and stack you want to use: clear the overflow bit to use the main memory; set the overflow bit (cause an overflow condition) to use the auxiliary memory.

After you have set up the parameters, pass control to the XFer routine by a jump instruction, rather than a subroutine call.

▲ **Warning** It is your responsibility as the programmer to save the current stack pointer before using XFer and to restore it after regaining control. Failure to do so will cause program errors. Refer to "Interrupts" in Chapter 3 for instructions on how to do this. ▲

Using display memory switches

Selection of main or auxiliary RAM for the 48K address space is described earlier in this chapter. However, under many circumstances your program may want to control reading and writing to display pages separately. The switches discussed in this section override the effects of RAMRd and RAMWrt for display pages only.

Three switches are involved in the display page selection process. Each of them has three locations assigned to it: one to turn it on, one to turn it off, and a third to read its state (see Table 2-6). One of the switches, 80Store, shares its on and off addresses with a keyboard reading function. As a result, your program must write to these locations to turn the switch on and off.

■ **Table 2-6** Display memory switches

Name	Action	Hex	Dec	Function
80Store	W	$C000	49152	Off: RAMRd and RAMWrt determine RAM locations.
80Store	W	$C001	49153	On: Page2 switches between TLP1 and TLP1X, and (if HiRes on) between HRP1 and HRP1X.
Rd80Store	R7	$C018	49176	Read whether 80Store on (1) or off (0).
Page2	R	$C054	49236	Off: Select TLP1 and HRP1.
Page2	R	$C055	49237	On: If 80Store off, switch to TLP2, and (if HiRes on) to HRP2. If 80Store on, switch to TLP1X, and (if HiRes on) to HRP1X.
RdPage2	R7	$C01C	49180	Read whether Page2 on (1) or off (0).
HiRes	R	$C056	49238	Off: Display text and Lo-Res graphics page.
HiRes	R	$C057	49239	On: Display Hi-Res pages; make Page2 switch between Hi-Res pages.
RdHiRes	R7	$C01D	49181	Read whether HiRes on (1) or off (0).
IOUDis	W	$C07E	49278	On: Disable IOU access for addresses $C058 to $C05F; enable access to DHiRes switch*.
IOUDis	W	$C07F	49279	Off: Enable IOU access for addresses $C058 to $C05F; disable access to DHiRes switch*.
RdIOUDis	R7	$C07E	49278	Read IOUDis switch (1=off)†.
DHiRes	R/W	$C05E	49246	On: (If IOUDis on) turn on Double Hi-Res.
DHiRes	R/W	$C05F	49247	Off: (If IOUDis on) turn off Double Hi-Res.
RdDHiRes	R7	$C07F	49279	Read DHiRes switch (1=off)†.

* The firmware normally leaves IOUDis on.

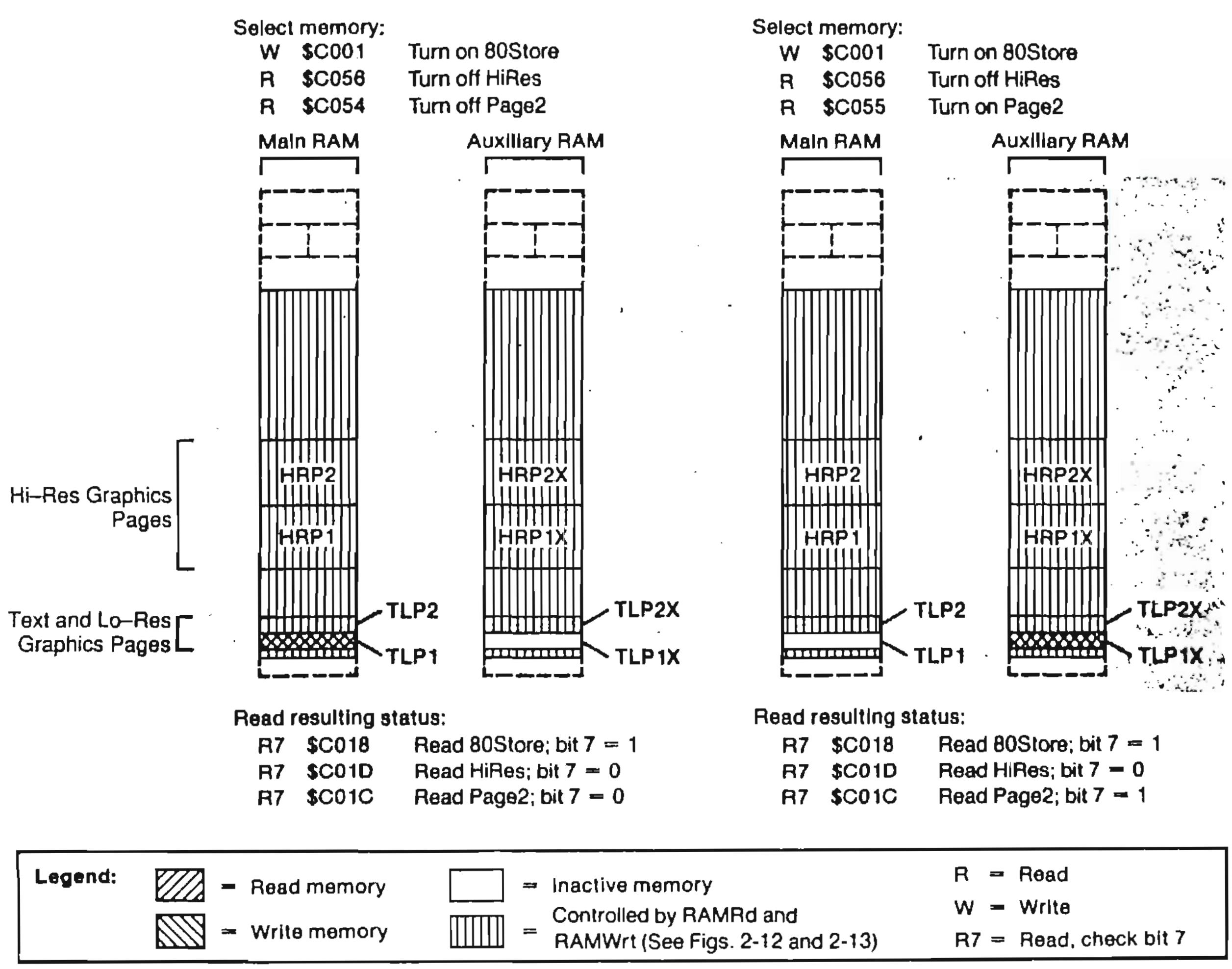
† Reading or writing any address in the range $C070–$C07F also triggers the paddle timer and resets the VBL interupt flag (see Chapter 9).

For each switch, you can read bit 7 at the third location to check whether the switch is on or off. If the switch is on, bit 7 is 1; if the switch is off, bit 7 is 0.

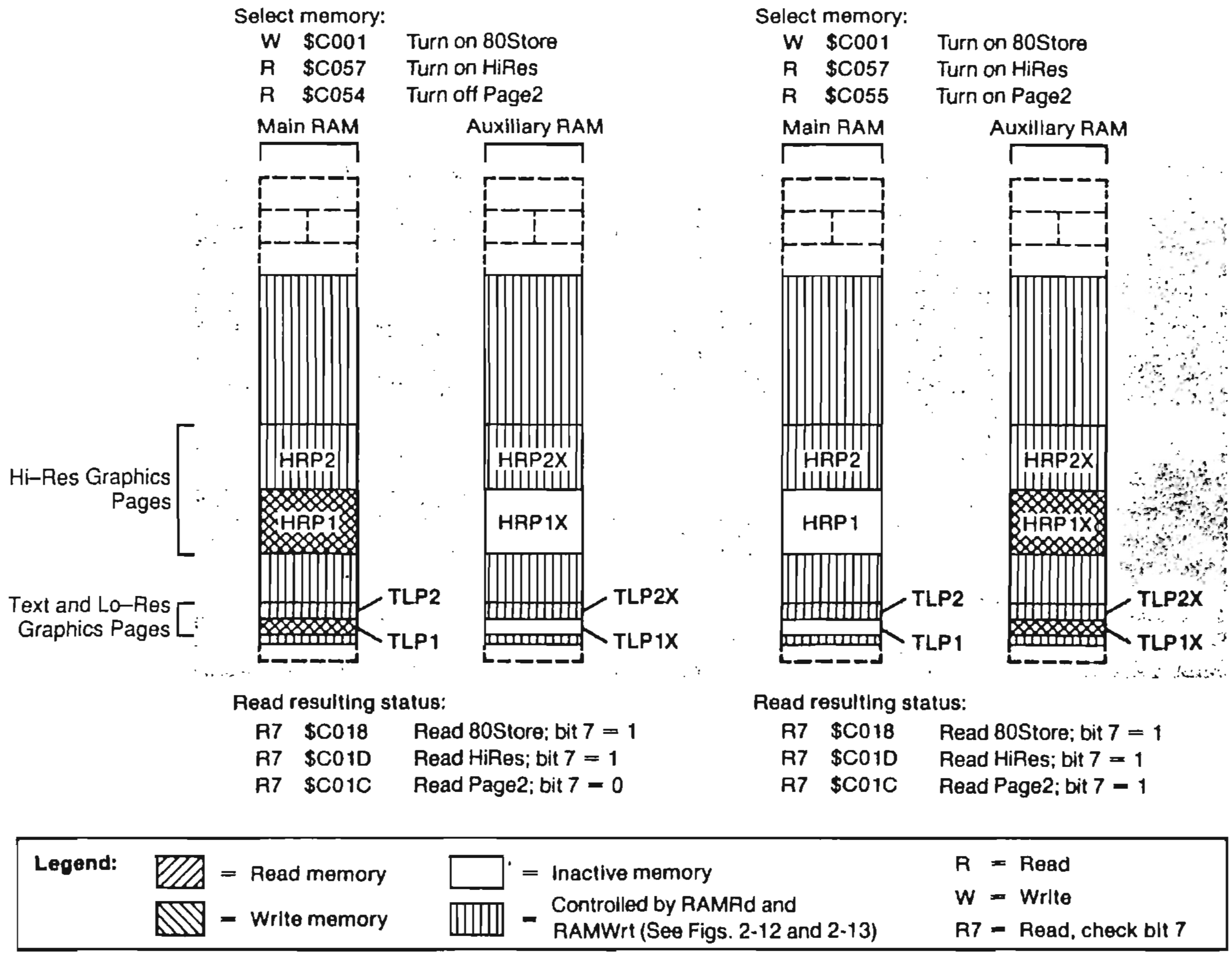
Here is how these switches work for reading and writing:

- If HiRes is off, then Page2 switches between text and Lo-Res graphics pages (TLP) only. If HiRes is on, then Page2 switches between TLP and Hi-Res graphics pages (HRP).
- If 80Store is off, RAMRd and RAMWrt (described in Table 2-2) determine whether main or auxiliary RAM locations are used. Page2 selects pages for display (as described in Chapter 6), but not for reading and writing.
- If 80Store is on, it overrides RAMRd and RAMWrt with respect to the display pages selected by HiRes and Page2 (see Figures 2-12 and 2-13).

Figure 2-12 Page2 selections, 80Store on and HiRes off



■ **Figure 2-13** Page2 selections, 80Store on and HiRes on



Chapter 3 **Resets and Interrupts**

This chapter describes resets and interrupts in the Apple IIc and Apple IIc Plus computers. The first half of the chapter explains how to start up and reset the computer and what happens in hardware and firmware when you do so. The second half of the chapter describes 65C02 interrupts, and explains how interrupt handlers are used on the Apple IIc–family computers. ■

Starts and resets

After the reset signal (RESET*) to the 65C02 processor is asserted, the processor goes through an initialization sequence, then loads into the program counter the address of a routine in ROM called the *system reset handler*. As shown in Figure 3-1, the system reset handler puts all hardware devices and soft switches into a known state, and then checks the reset vector. The address of the system reset handler is stored at locations $FFFC and $FFFD in ROM. The system reset handler is located at $FA62. Table 3-1 shows the address of the reset vector and of other vectors located in page $03.

▲ **Warning** All of the firmware in the Apple IIc–family computers assumes that, after a reset, the system is in the state set by the system reset handler. If you change the pointer at $FFFC and $FFFD to point to your own reset handler, the system will almost certainly crash. ▲

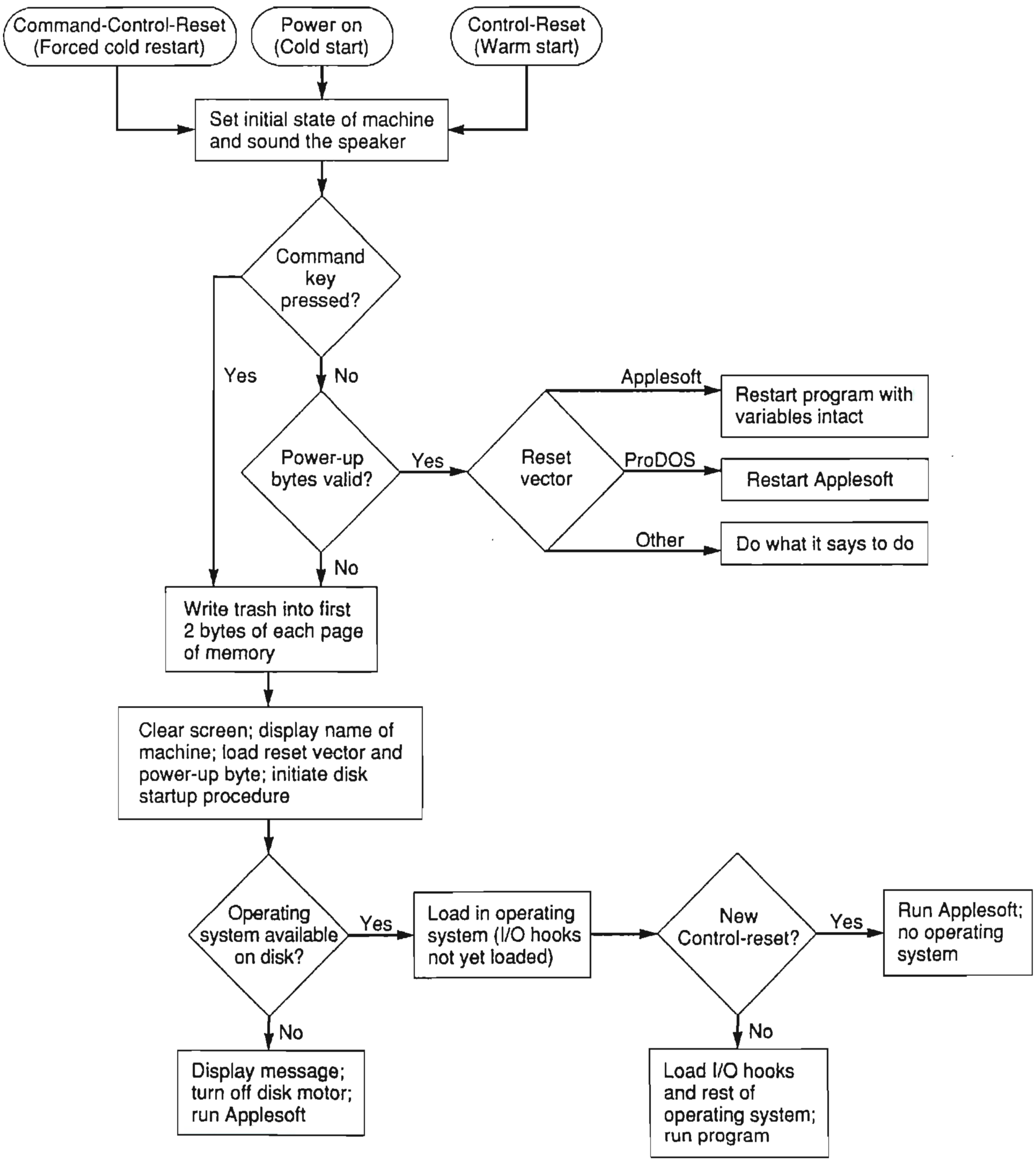
If the reset vector is valid, the system reset handler passes control to the program pointed to by the reset vector. If the reset vector is not valid, the reset routine attempts to load an operating system, and then passes control to that operating system.

There are three ways to reset the Apple IIc or Apple IIc Plus:

- power-on reset (cold start)
- forced cold-start reset (forced cold reset)
- warm-start reset (warm start)

Each of these methods calls the reset routine.

■ **Figure 3-1** Reset routine flowchart



■ **Table 3-1** Page $03 vectors

Vector address	Vector function
$03F0 (1008) $03F1 (1009)	Address of the subroutine that handles BRK instructions (normally $FA59)
$03F2 (1010) $03F3 (1011)	Reset vector (see text)
$03F4 (1012)	Validity-Check byte (see text)
$03F5 (1013)	Jump instruction to the subroutine that handles Applesoft & commands (normally $4C $58 $FF)
$03F6 (1014)	
$03F7 (1015)	
$03F8 (1016)	Jump instruction to the subroutine that handles user Control-Y commands
$03F9 (1017)	
$03FA (1018)	
$03FB (1019)	Not used in the Apple IIc family (contains jump instruction to the subroutine that handles nonmaskable interrupts)
$03FC (1020)	
$03FD (1021)	
$03FE (1022)	Interrupt vector (address of the subroutine that handles interrupt requests)
$03FF (1023)	

The initial state of the machine

When you initiate a reset, hardware and firmware in the Apple IIc family set the machine to the following initial state:

- main memory ROM and RAM are enabled
- auxiliary RAM is disabled
- bank-switched memory space is set to read from ROM and write to RAM
- main RAM Bank 2 is enabled

Next, the reset routine performs the following steps:

1. It displays the contents of 40-column text Page 1 using the primary character set.
2. It sets the display window equal to the full 40-column display.
3. It puts the cursor at the bottom of the screen.
4. It sets the text display format to normal.
5. It sets the keyboard as the standard input device.
6. It sets the display as the standard output device.
7. It masks mouse interrupts.
8. It sets mouse defaults (described in Chapter 9).
9. It turns on the IOUDis memory switch, which enables access to the DHiRes switch.
10. It sets DHiRes off.
11. In the Apple IIc Plus only, it sets ports 1, 2, 5, and 6 to operate at 1 MHz and ports 3, 4, and 7 to operate at 4 MHz (see following note).
12. It clears the keyboard strobe.
13. It beeps the speaker.

△ Apple IIc Plus When the Apple IIc Plus computer executes the restart procedure, the processor is set to run at 4 MHz, unless the user holds down the Esc key while resetting the computer. If the user holds down the Esc key while pressing Control-Command-Reset or pressing Control-Reset, then the Apple IIc Plus is set to run at 1.023 MHz (the same speed as other Apple IIc computers). The Apple IIc Plus accelerator feature is discussed in more detail in Chapter 11. △

Memory selector soft switches and the various address maps used by the Apple IIc family are described in Chapter 2. Standard I/O devices are described in Chapter 4. Mouse defaults are described in Chapter 9.

With the computer in a known state, the reset routine uses the Validity-Check byte at location $03F4 to determine if the reset vector is valid, as described in the next section.

The reset vector

The reset routine determines from the reset vector which application to load upon reset. The reset vector is valid only after a cold reset has already been completed. If the reset vector is valid, the computer performs the warm-start procedure; if the reset vector is not valid, the computer performs the cold-start procedure. The reset routine uses the Validity-Check byte and the high-order byte of the reset vector (together referred to as the *power-up bytes*) to determine if the reset vector is valid.

When the cold-reset routine, operating system, or application sets a reset vector, it also sets the Validity-Check byte by performing an exclusive OR (XOR) operation on the high-order byte of the reset vector (at $03F3) and the constant 165 (hexadecimal $A5).

The reset routine checks the reset vector for validity by performing an XOR operation on the high-order byte of the reset vector and the constant 165. The result of this XOR operation is compared with the value stored in the Validity-Check byte. If the values match, the reset routine considers the vector to be valid and procedes with the warm-start procedure. If they do not match, the reset routine considers the vector to be not valid and procedes with the cold-start procedure.

◆ *Note:* You can change the reset vector so that the reset routine transfers control to your program (the *user reset handler*) when the user performs a warm-start. For this to work, you must also change the Validity-Check byte to equal the result of performing an operation XOR on the high-order byte of your new reset vector and the constant 165 ($A5). If you fail to do this, then the next time the user resets the Apple IIc or Apple IIc Plus, the reset routine will determine that the reset vector is not valid and will execute the cold start procedure, eventually transferring control to the disk bootstrap routine or to Applesoft.

You can use the SetPwrC subroutine to generate the Validity-Check byte for the current reset vector. This subroutine is at location $FB6F. When your program finishes, it can return the Apple IIc or Apple IIc Plus to normal operation by restoring the original reset vector and again calling the SetPWRC subroutine to set the Validity-Check byte.

The cold-start procedure

The cold-start procedure can be initiated in one of two ways:

- power-up reset (turning on your Apple IIc or Apple IIc Plus)
- forced cold reset (see "Forced Cold Reset," later in this chapter)

A cold start is executed whenever a reset has been initiated and the Validity-Check byte is not valid. With the computer in a known state, the reset routine clears the display and puts the string `Apple IIc` or `Apple IIc +` at the top of the display. It then loads $03F2 and $03F3 (the reset vector) with the Applesoft interpreter entry point address and calculates the Validity-Check byte, loading $03F4 with the result.

With the Validity-Check byte loaded, the reset routine attempts to load an operating system from a startup device. The reset routine passes control to the entry point for the device that has the highest priority. Block 0 of the device is read into location $0800 in main memory. If location $0800 contains $01 and location $0801 is nonzero, the device is considered to contain valid startup code.

If there is valid startup code in the device, control is passed to code starting at location $0801, and the program or operating system code loaded begins running. If there is no valid startup code in the device, control is returned to the reset routine. The reset routine then tries the entry point for the device with the next highest priority, following exactly the same steps.

This procedure is repeated until valid startup code is found or until all devices capable of booting the computer have been tried. In the event no valid startup code is found, the reset routine prints the message `Check Disk Drive` at the bottom of the screen.

△ Apple IIc Plus — The Apple IIc Plus computer reset routine prints the message `Unable to Find a Bootable Disk Online` at the bottom of the screen. △

◆ *Note:* If you press Control-Reset before the startup procedure is completed, the reset routine passes control to the Applesoft BASIC interpreter.

The priorities assigned to startup devices, and the boot-device entry points, depend on the version of the Apple IIc. The startup configurations for all Apple IIc family members are given in the following sections.

△ Important DOS and versions of Pascal earlier than 1.3 do not support the multiple-device startup protocol; you must use ProDOS, Pascal 1.3 (or later), or a similar operating system to take advantage of this protocol. △

To start up the computer from a specific disk drive rather than using the reset handler's priority scheme, use one of the following methods:

- issue a PR#*n* command from BASIC to jump to $Cn00
- issue a *n* Control-P command from the Monitor to jump to $Cn00
- execute a JMP instruction in a program to jump to $Cn00

The number *n* is the port number of any device that contains valid startup code; for example, PR#6 causes the Apple IIc to boot from the internal 5.25-inch disk drive and causes the Apple IIc Plus to boot from the first external 5.25-inch disk drive. The memory expansion Apple IIc and the Apple IIc Plus can be booted from a properly formatted memory expansion card by using 4 for *n*.

When you turn on the computer or press Command-Control-Reset, the computer attempts to start up from the highest-priority device, then tries the other startup devices in a preset sequence. When you jump to a particular port, on the other hand, the computer attempts to start up from the device at that port only. If the computer fails to find valid boot code at the port you specified, it displays an error message on the screen and passes control to the Applesoft interpreter in ROM.

Original Apple IIc

The original Apple IIc has one internal 5.25-inch disk drive and supports one external 5.25-inch drive. Only the internal 5.25-inch disk drive can be used as a startup device for a cold start. The internal disk drive entry point is $C600 in the main ROM (port 6, drive 1).

The routines at $C600 turn on the drive motor, recalibrate the read/write head at track 0, and read sector 0. The data read from track 0, sector 0 of the startup disk is then loaded and executed.

◆ *Note:* It is possible to load the original Apple IIc from the external 5.25-inch drive, but it can only be done from either the Monitor or BASIC. The procedures are as follows:
  - from the Monitor, press 7 Control-P
  - from Applesoft, type `PR#7`

UniDisk 3.5 Apple IIc

The UniDisk 3.5 Apple IIc has one internal 5.25-inch disk drive, and supports one external 5.25-inch drive and one external 3.5-inch disk drive. The possible startup devices for the UniDisk 3.5 Apple IIc are, in order of priority, as follows:

- internal 5.25-inch disk drive at $C600 (port 6, drive 1)
- external 3.5-inch disk drive at $C500 (port 5, drive 1)

All addresses are for the main ROM.

The routines at $C500 and at $C600 turn on the drive motor, recalibrate the read/write head at track 0, and read sector 0. The data read from track 0, sector 0 of the startup disk is then loaded and executed.

◆ *Note:* The UniDisk 3.5 Apple IIc cannot start up from the external 5.25-inch drive (port 6, drive 2).

Memory expansion Apple IIc

The memory expansion Apple IIc has one internal 5.25-inch disk drive, and supports up to three external devices plus a memory expansion card. The external devices can be any combination of up to three external 3.5-inch disk drives and one external 5.25-inch disk drive. The possible startup devices for the memory expansion Apple IIc are, in order of priority, as follows:

- memory expansion card at $C400 (port 4, drive 1)
- internal 5.25-inch disk drive at $C600 (port 6, drive 1)
- external UniDisk 3.5 drive at $C500 (port 5, drive 1)

All addresses are for the main ROM.

The routines at $C400 call the SmartPort Read Block routine. The Read Block routine reads block $00 from the memory expansion card and loads it into main memory.

△ Important — A memory expansion card cannot start up the memory expansion Apple IIc after a power-up reset, as the contents of the expansion RAM are erased when the power is switched off. Memory expansion cards are discussed in the section "Memory Expansion Card," in Chapter 11. △

The routines at $C500 and at $C600 turn on the drive motor, recalibrate the read/write head at track 0, and read sector 0. The data read from track 0, sector 0 of the startup disk is then loaded and executed.

◆ *Note:* The memory expansion Apple IIc cannot start up from either the external 5.25-inch disk drives (port 6, drive 2 and port 2 drive 1) or the second external 3.5-inch disk drive (port 5, drive 2).

Apple IIc Plus

The Apple IIc Plus has one internal 3.5-inch disk drive, and supports up to three external drives plus the memory expansion card. The external drives can be any combination of one Apple 3.5 drive, up to three UniDisk 3.5 drives, and up to two 5.25-inch disk drives. The possible startup devices for the Apple IIc Plus are, in order of priority, as follows:

- memory expansion card at $C400 (port 4, drive 1)
- internal 3.5-inch drive at $C500 (port 5, drive 1)
- external UniDisk 3.5 or Apple 3.5 drive at $C500 (port 5, drive 2)
- external 5.25-inch drive at $C600 (port 6, drive 1)

All addresses are for the main ROM.

◆ *Note:* The hardware and firmware of the Apple IIc Plus computer can support up to five disk drives, but the power supply supports only four (counting the internal disk drive).

The routine at $C400 calls the SmartPort Read Block routine. The Read Block routine reads block $00 from the memory expansion card and loads it into main memory.

△ Important — The memory expansion card cannot start up the Apple IIc Plus after a power-up reset, as the contents of the expansion RAM are erased when the power is switched off. △

The routines at $C500 and $C600 turn on the drive motor, recalibrate the read/write head at track 0, and read sector 0. The data read from track 0, sector 0 of the startup disk is then loaded and executed.

◆ *Note:* The Apple IIc Plus cannot start up from the second external 5.25-inch disk drive (port 6, drive 2), nor can it startup from the second or third external 3.5-inch disk drives (port 2, drive 1 and port 2, drive 2).

Forced cold reset

You can initiate the forced cold reset procedure by pressing Control-Command-Reset. A forced cold reset is a way to force the Apple IIc or Apple IIc Plus to execute a cold start without turning the computer off and back on. A forced cold reset is useful if you want to terminate a program unconditionally or bail out of some sort of programming problem.

◆ *Note:* On all Apple IIc family members but the original Apple IIc, you must hold the Command key (Open Apple on older Apple IIc computers) down until the internal drive starts to spin. If you release the Command key before the drive starts to spin, the computer drops into BASIC instead of rebooting.

In addition to the normal initialization performed by all resets, a forced cold reset "destroys" the contents of main memory by writing 2 bytes of meaningless data into the beginning of each page of main RAM. The reset vector at $03F2 and $03F3 is destroyed in the process. Then the reset routine loads the reset vector with the Applesoft interpreter entry point address and calculates the Validity-Check byte, loading $03F4 with the result.

With the Validity-Check byte loaded, the reset routine attempts to load an operating system from a startup device, just as for a power-on reset. See the section "The Cold-Start Procedure," earlier in this chapter, for details.

△ Important Whenever possible, you should use a forced cold reset to terminate processing instead of turning the power off and on; it is much less stressful to the computer hardware. (Notice, however, that if you have a memory expansion card configured as a startup disk, the only way to achieve a complete reset and avoid restarting from the memory expansion card is to turn the power off and back on.) △

◆ *Note:* If you press both the Command key and the Option key (the Solid Apple key on older Apple IIc computers) during power-up or a forced cold restart, built-in diagnostic code is executed.

The warm-start procedure

A warm start is executed whenever a reset has been initiated and the Validity-Check byte is valid. With the computer in a known state, the reset routine passes control to the routine pointed to by the user reset vector, which points to the controlling program or operating system. The controlling program restarts and operates normally. What the controlling program or operating system does when it restarts depends on the specific program or operating system.

As long as nothing has been done to alter the reset vector or the Validity-Check byte since power-up, the reset vector is still considered valid, and it points to the Applesoft interpreter. The reset routine passes control to Applesoft through the reset vector and Applesoft restarts your program, with your variables intact.

If the reset vector has been altered by an application program or operating system, the Validity-Check byte must also have been altered to match the new vector address. If the Validity-Check byte has not been set correctly, then the validity check fails and the reset routine initiates the cold-start procedure.

If you are using DOS or ProDOS, the reset vector has been altered to point to the operating system in use and the Validity-Check byte has been altered to match. In this case, the reset routine passes control through the altered reset vector to the operating system in use at the beginning of the reset procedure.

The user can initiate a warm reset by pressing Control-Reset. An application program or operating system can initiate a reset by jumping to the reset routine. The vector to the reset routine is at locations $FFFC and $FFFD in ROM; this vector normally points to the system reset handler at $FA62.

△ Important A program residing only in bank-switched RAM cannot use the reset vector to regain control after a reset because upon reset the hardware selects the ROM for reading in the bank-switched memory space. △

Interrupts

An **interrupt** is a signal to the processor that indicates that the computer should stop whatever it's doing so it can quickly handle a time-dependent task. For example, the Apple IIc mouse sends an interrupt to the computer every time it moves. This lets the system keep track of the mouse's position and maintain smooth movement of the pointer on the screen.

▲ **Warning** If you use interrupt hardware directly, instead of using the built-in interrupt-handling firmware, you can't be sure that your programs will be compatible with possible future Apple II–series computers or revisions. ▲

Introduction

When an interrupt occurs, control passes to an interrupt handler, which must record the exact state of the computer at the moment of the interrupt, determine the source of the interrupt, and take appropriate action. It is important that the computer preserve a "snapshot" of its state when interrupted, so that when it continues later with what it was doing, those conditions can be restored.

Interrupts have not always been fully supported on the Apple II. All versions of Apple's DOS, as well as the Monitor program, rely on the integrity of location $45; however, the built-in interrupt handler in the Apple II Plus and original Apple IIe computers destroys the contents of this location by saving the A register (accumulator) in it. Most versions of Pascal do not work on these machines with interrupts enabled.

The Apple IIc–family built-in interrupt handler saves the A register (accumulator) on the stack instead of in location $45. DOS 3.3, ProDOS, Pascal 1.2 (or later versions), and the Monitor all work with interrupts on the Apple IIc family.

△ Important Because interrupt requests that occur while interrupts are disabled cannot be detected, interrupts are effective only if they are enabled most of the time. Because of the critical timing of disk read and write operations, Pascal, DOS 3.3, and ProDOS disable interrupts while accessing the disk. Thus it is important to remember that while a disk drive is being accessed, all the interrupts discussed in this chapter are disabled. △

Interrupt handling on the 65C02

From the point of view of the 65C02, there are three possible causes of interrupts.

1. The IRQ' line on the microprocessor can be pulled low. If 65C02 interrupts are not masked (that is, the CLI instruction has been used), the CPU can respond to an IRQ interrupt. This is the standard technique that devices use when they need immediate attention.
2. The processor executes a break instruction (BRK, opcode $00).
3. A nonmaskable interrupt (NMI) occurs. Because the NMI' line in the 65C02 of the Apple IIc and Apple IIc Plus computers is not used, this never happens.

The first two possibilities cause the 65C02 to save the current program counter and status byte on the stack and then jump to the routine whose address is stored in $FFFE and $FFFF. The sequence performed by the 65C02 is as follows:

1. If an IRQ interrupt occurs, finish executing the current instruction.
2. Push the high byte of the program counter onto the stack.
3. Push the low byte of the program counter onto the stack.

4. Push the program status byte onto the stack.

5. Jump to the address stored in $FFFE, $FFFF—that is, JMP ($FFFE).

The different sources of interrupt signals are discussed below.

The interrupt vector at $FFFE

In the Apple IIc family there are three separate regions of memory that can be accessed by address $FFFE, depending on how the soft switches are set:

- the built-in ROM
- the bank-switched memory in main RAM
- the bank-switched memory in auxiliary RAM

The vector at $FFFE in the ROM points to the built-in interrupt handling routine. You should use the built-in interrupt handler, rather than writing your own, because of the complexity of interrupts in the Apple IIc family.

When you initialize the mouse or serial communication firmware, copies of the ROM's interrupt vector are placed in the interrupt vector addresses in both main and auxiliary bank-switched memory. If you plan to use interrupts and the bank-switched memory without the mouse or communication firmware, you must copy the ROM's interrupt vector yourself.

The built-in interrupt handler

The built-in interrupt handler is responsible for determining whether a BRK or an IRQ interrupt occurred. If it was an IRQ interrupt, the handler decides whether the interrupt should be handled internally, handled by the user, or simply ignored.

The handler records the current memory configuration, then sets up its own standard memory configuration so that a user's interrupt handler knows the precise memory configuration when it is called.

Next the handler checks to see if the interrupt was caused by a break instruction, and if it was, handles it as described later in this chapter.

If the interrupt was not caused by a BRK instruction, the handler checks for interrupts that it knows how to handle (for example, a properly initialized mouse) and handles them.

Depending on the state of the system, it either ignores other interrupts or passes them to a user's interrupt-handling routine whose address is stored at $03FE and $03FF in main memory.

After handling an interrupt itself, or after the user's handler returns an RTI instruction, the built-in handler restores the memory configuration, and then executes an RTI instruction to reactivate processing. Table 3-2 illustrates this whole process. Each of the steps is explained in detail in the sections that follow.

■ **Table 3-2** Interrupt-handling sequence

Interrupted program	**Processor**	**Built-in handler**	**User's handler**
Program →	Push address Push status		
	JMP ($FFFE) →	Save old and set new memory configuration	
		If BRK, then go to break handler ($FA47) →	
		Our interrupt?	
		NO: Push address	
		Push status	
		JMP ($03FE) →	Handle interrupt
			. . .
		YES: Handle it	. . .
		Restore memory configuration ←	RTI
	Pull status ←	RTI	
Program ←	Pull address		

Saving the memory configuration

The built-in interrupt handler saves the state of the system and sets it to a known state according to these rules:

- If 80Store and Page2 are on, then it switches in text Page 1 (Page2 off) so that main screen holes are accessible.
- It switches in main memory for reading (RAMRd off).
- It switches in main memory for writing (RAMWrt off).
- It switches in ROM addresses \$D000–\$FFFF for reading (RdLCRAM off).
- It switches in main stack and zero page (AltZP off).
- It preserves the auxiliary stack pointer and restores the main stack pointer.
- It preserves the current ROM state and switches in the ROM bank 1.

△ Important Because main memory is switched in, all memory addresses used later in this chapter are in main memory unless otherwise specified. △

Managing main and auxiliary stacks

Because the Apple IIc family has two stack pages, the firmware has established a convention that allows the system to be run with two separate stack pointers. Two bytes in the auxiliary stack page are to be used as storage for inactive stack pointers: \$0100 for the main stack pointer when the auxiliary stack is active, and \$0101 for the auxiliary stack pointer when the main stack is active.

When a program that uses interrupts switches in the auxiliary stack for the first time, it should place the value of the main stack pointer at auxiliary stack address \$0100, and initialize the auxiliary stack pointer to \$FF (the top of the stack). When it subsequently switches from one stack to the other, it should save the current stack pointer before loading the pointer for the other stack.

When an interrupt occurs while the auxiliary stack is switched in, the current stack pointer is stored at \$0101, and the main stack pointer is retrieved from \$0100. Then the main stack is switched in for use. After the interrupt has been handled, the stack pointer is restored to its original value.

User's interrupt handler at $03FE

You can set up screen hole locations to indicate that the user's interrupt handler should be called when certain interrupts occur. To do this, place your interrupt handler's address at $03FE and $03FF in main memory, low byte first.

The user's interrupt handler should do the following:

- Verify that the interrupt came from the expected source. The following sections describe how this should be done for each interrupt source.
- Handle the interrupt as desired.
- Clear the interrupt, if necessary. The following sections describe how to clear the interrupts.
- Return with an RTI instruction.

If your interrupt handler needs to know the memory configuration at the time of the interrupt, it can check the encoded byte stored 4 bytes down on the stack. This byte is explained later in this chapter.

In general there is no guaranteed *maximum* response time for interrupts. This is because the system may be doing a disk operation, which could last for several seconds.

Once the built-in interrupt handler has been called, it takes about 250 to 300 microseconds for it to call your interrupt-handling routine. After your routine returns, it takes 40 to 140 microseconds to restore memory and return to the interrupted program.

If memory is in the standard state when the interrupt occurs, the total overhead for interrupt processing is about 150 microseconds less than if memory is in the worst possible state (80Store and Page2 on, auxiliary memory switched in for reading and writing, auxiliary bank-switched memory page $02 switched in for reading and writing).

Handling break instructions

After the interrupt handler has set the memory configuration, it checks to see if the interrupt was caused by a BRK (opcode $00) instruction. (If it was, bit 4 of the processor status byte is a 1.) If so, it jumps to a break-handling routine, which saves the state of the computer at the time of the break as shown in Table 3-3. The break routine then jumps to the routine whose address is stored at $03F0 and $03F1.

■ **Table 3-3** Break handler state-saving format

Information	Location
Program counter (low byte)	$3A
Program counter (high byte)	$3B
Encoded memory state	$44
Accumulator	$45
X register	$46
Y register	$47
Status register	$48

The encoded memory state in location $44 is defined as shown in Table 3-4.

■ **Table 3-4** Encoded memory state definition

Bit	Definition
7	1 if alternate zero page and stack switched in
6	1 if 80Store and Page2 both on
5	1 if auxiliary RAM switched in for reading
4	1 if auxiliary RAM switched in for writing
3	1 if bank-switched RAM being read
2	1 if bank $01 of page $D000 switched in
1	1 if bank $02 of page $D000 switched in
0	1 if auxiliary ROM switched in

Sources of interrupts

The CPU can receive interrupts from many different sources. Each source is enabled and used slightly differently from the others. There are two basic sources of interrupts: use of the mouse and actions affecting the two 6551 ACIA circuits (the chips that control serial communication). How to use these sources of interrupts in conjunction with the built-in interrupt handler is discussed later in this chapter.

Mouse use can cause interrupts when

- the mouse is moved in the horizontal (X) direction
- the mouse is moved in the vertical (Y) direction
- the mouse button is pressed

Interrupts can also be generated every 1/60 second by the rising edge of the vertical blanking signal (VBL). This is called the *vertical blanking interrupt* (VBL interrupt) and is synchronized with a signal used for the video display.

Actions affecting the ACIA circuits can cause interrupts when

- a key is pressed (the firmware can use this interrupt to buffer keystrokes, or it can pass the interrupt on to the user)
- either ACIA has received a byte of data from its port (the firmware can use this interrupt to buffer data or it can pass the interrupt on to the user)
- pin 5 of either serial port changes state (device ready/not ready to accept data) (when the serial firmware is active, this interrupt is absorbed; however, the serial firmware uses the signal to decide whether or not to transmit the next byte of data)
- either ACIA is ready to accept another character to be transmitted (when the serial firmware is active, this interrupt is absorbed; however, the serial firmware uses the signal to decide whether or not to transmit the next byte of data)
- the keyboard strobe is cleared (the firmware absorbs this interrupt)

An interrupt can also be generated by a device attached to the external disk drive port. The firmware can pass this interrupt on to the user.

Firmware handling of interrupts

The following sections discuss how the various sources of interrupts should be used together with the built-in interrupt handler.

Firmware for mouse and VBL interrupts

As described in Chapter 9, the mouse can be initialized (by the SetMouse call) to eight different modes that enable one or more sources of interrupts. In transparent mode, the interrupts are entirely handled by the built-in interrupt handler; the other modes require a user-installed interrupt handler.

△ Apple IIc Plus   Using the mouse in vertical blanking active modes causes the Apple IIc Plus computer to run at 1 MHz regardless of the speed set at power-up or reset. When you use the mouse in transparent mode, on the other hand, the Apple IIc Plus can run at 4 MHz. △

When the mouse is initialized, the interrupt vector is copied to addresses $FFFE and $FFFF in main and auxiliary bank-switched RAM. This permits mouse interrupts with any memory configuration.

When the mouse is active, possible sources of interrupts are those listed earlier in this chapter as resulting from mouse use.

When an interrupt occurs, the built-in interrupt handler determines whether that particular interrupt source was enabled (by the SetMouse call). If so, the user's interrupt handler, whose address is stored at $03FE, is called.

The user's interrupt handler should first call ServeMouse to determine the source of the interrupt. This call updates the mouse status byte at $077C and returns with the carry bit clear if mouse movement, button, or vertical blanking was the source of the interrupt.

The values of this mouse status byte at $077C ($077F in the memory expansion Apple IIc) are as follows:

Bit	1 Means that
3	Interrupt was from vertical blanking
2	Interrupt was from button
1	Interrupt was from mouse movement

If the interrupt was due to mouse movement or button, the user's interrupt handler should then do a call to ReadMouse. This causes the mouse coordinates and status to be updated as follows:

$047C	Low byte of X coordinate
$04FC	Low byte of Y coordinate
$057C	High byte of X coordinate
$05FC	High byte of Y coordinate
$077C	Button and movement status

Bit	Means
7	0 = button up; 1 = button down
6	0 = button up on last ReadMouse 1 = button down on last ReadMouse
5	0 = no movement since last ReadMouse 1 = movement since last ReadMouse
3-1	Always set to 0 (interrupt cleared)

After the interrupt has been handled, the routine should terminate with an RTI instruction.

Remember that interrupts may be missed during disk accesses.

If you turn on mouse interrupts without initializing the mouse, the built-in interrupt handler takes care of the interrupts and your program never knows they occurred. If you want to handle mouse interrupts yourself, you must write your own interrupt handler and place vectors to it at addresses $FFFE and $FFFF in bank-switched RAM, and set bit 7 of the mouse port mode byte. See the section "Movement Interrupt Mode," in Chapter 9, for more information on this feature. Interrupts will be ignored whenever the $D000-$FFFF ROM is switched in.

Firmware for keyboard interrupts

The Apple IIc–family hardware is able to generate an interrupt when a key is pressed. The firmware is able to buffer up to 128 keystrokes, completely transparently, when properly enabled to do so. It saves them in the second half of page $08 of auxiliary memory. After the buffer is full, subsequent keystrokes are ignored. Because interrupts are only generated when keypresses occur, characters generated by the auto-repeat feature are not buffered. They can, however, be read when the buffer is empty.

Once keyboard buffering has been turned on, the next key should be read by calling RdKey ($FD0C).

▲ **Warning** Do not call the buffer-reading routine directly. Its entry address will not be the same in future versions of the computer. ▲

The special characters Control-S (stop list) and Control-C (stop Applesoft execution) do not work while keyboard buffering is turned on. Use Option-Control-X to clear the buffer.

Using keyboard buffering firmware

Keyboard buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself this way:

1. Disable processor interrupts (SEI).
2. Set location $05FA to $80. This tells the firmware to buffer keystrokes without calling the user's interrupt handler.
3. Set locations $05FF and $06FF to $80. These are pointers to where in the buffer the next keystroke will be stored and where the next will be read from, respectively.
4. Turn on the ACIA for port 2 by setting the low nibble of $C0AA to the value $0F. For example:

   ```
   LDA $C0AA     Read port 2 ACIA command register
   ORA #$0F      Set low nibble to $0F
   STA $C0AA     Set port 2 ACIA command register
   ```

   If you are using the serial ports at the same time, just set the low bit of $C0AA to 1. This prevents receiver interrupts from being turned off.

   A PR#2 or IN#2 command or the equivalent shuts off keyboard interrupts.
5. Enable processor interrupts (CLI).

△ Apple IIc Plus Memory expansion The memory expansion Apple IIc and the Apple IIc Plus use some of the screen holes differently than earlier Apple IIc computers. For the memory expansion Apple IIc and Apple IIc Plus, change all $nnnF addresses to $nnnC (for example, change $05FF to $05FC). △

Using keyboard interrupts through firmware

Keyboard interrupts are received through the ACIA for port 2. They can be enabled as follows:

1. Disable processor interrupts (SEI).
2. Set location $05FA to $C0. This tells the firmware to identify a keystroke interrupt and to call the user's interrupt handler.
3. Turn on the ACIA for port 2 by setting the low nibble of $C0AA to the value $0F. For example:

```
LDA $C0AA     Read port 2 ACIA command register
ORA #$0F      Set low nibble to $0F
STA $C0AA     Set port 2 ACIA command register
```

4. Enable processor interrupts (CLI).

When the user's interrupt handler is called, it can identify the keyboard as the interrupt source by reading location $04FA. This is a copy of the ACIA status register at the time of the interrupt. If the interrupt was due to something on the ACIA for port 2, bit 7 is set. If the interrupt was caused by a keystroke, bit 6 is set and bit 5 is unchanged.

After servicing this interrupt, the interrupt handler should clear the interrupt by setting $04FA to $00.

Using external interrupts through firmware

Pin 9 of the external disk drive connector (EXTINT) can be used to generate interrupts through the ACIA for port 1.

△ Apple IIc Plus — The EXTINT signal is not available on pin 9 of the Apple IIc Plus computer. The external disk drive connectors are described in the section "Disk I/O," in Chapter 11. △

The EXTINT signal can be used as a source of interrupts (on a high-to-low transition) if enabled as follows:

1. Disable processor interrupts (SEI).
2. Set location $05F9 to $C0. This tells the firmware to identify an external interrupt, and to call the user's interrupt handler.
3. Turn on the ACIA for port 1 by setting the low nibble of $C09A to the value $0F. For example:

```
LDA $C09A     Read port 1 ACIA command register
ORA #$0F      Set low nibble to $0F
STA $C09A     Set port 1 ACIA command register
```

4. Enable processor interrupts (CLI).

When the user's interrupt handler is called, it can identify this interrupt by reading location $04F9. This is a copy of the ACIA status register at the time of the interrupt. If the interrupt was due to something on the ACIA for port 1, bit 7 is set. If the interrupt was caused by the external interrupt line, bit 6 is clear and bit 5 is unchanged.

After servicing this interrupt, the interrupt handler should clear the interrupt by setting $04F9 to $00.

Firmware for serial interrupts

The Apple IIc–family hardware is able to generate interrupts both when the ACIA receives data and when it is ready to send data. The built-in interrupt handler responds to incoming data only. The firmware is able to buffer up to 128 incoming bytes of serial data from either serial port. After the buffer is full, subsequent bytes are ignored. Only one port can be buffered at a time. The following sections assume that the serial port to be buffered is already initialized, as explained in Chapter 7.

Using serial buffering transparently

Serial buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself, as follows:

1. Disable processor interrupts (SEI).
2. Set location $04FF to $C1 to buffer port 1, or to $C2 to buffer port 2.
3. Set locations $057F and $067F to $00. These are pointers to the next byte in the buffer to be used and the next character to be read from the buffer, respectively.
4. Turn on the ACIA for the port by setting the low nibble of $C09A for port 1 or $C0AA for port 2 to $0D. For example:

```
LDA $C09A    Read port 1 ACIA command register
AND $F0      Clear low nibble
ORA #$0D     Set low nibble to $0D
STA $C09A    Set port 1 ACIA command register
```

   The 0 in bit 1 of the command register enables receiver interrupts; thus an interrupt is generated when a byte of data is received.

5. Enable processor interrupts (CLI).

△ **Memory expansion** For the memory expansion Apple IIc, change all $nnnF addresses to $nnnC and change $0D to $09. △

When serial port buffering is thus enabled, normal reads from the serial port firmware fetch data from the buffer rather than directly from the ACIA.

Using serial interrupts through firmware

It is also possible to use the firmware to call the user interrupt handler whenever a byte of data is read by the ACIA. In this mode buffering is not performed by the firmware.

1. Disable processor interrupts (SEI).
2. Set location $04FF to a value other than $C1 or $C2.
3. Turn on the ACIA for the port by setting the low nibble of $C09A for port 1 or $C0AA for port 2 to $0D. For example:

`LDA $C09A`	Read port 1 ACIA command register
`AND $F0`	Clear low nibble
`ORA #$0D`	Set low nibble to $0D
`STA $C09A`	Set port 1 ACIA command register

   The 0 in bit 1 of the command register enables receiver interrupts; thus an interrupt is generated when a byte of data is received.
4. Enable processor interrupts (CLI).

△ Memory expansion — For the memory expansion Apple IIc, change all $nnnF addresses to $nnnC and change $0D to $09. △

When a serial port is thus enabled, the user's interrupt handler is called each time the port receives a byte of data. The status byte saved by the firmware ($04F9 for port 1; $04FA for port 2) has the high bit set if the interrupt occurred on that port. Bit 3 is set if the interrupt was due to a received byte of data.

The interrupt handler should clear the interrupt by clearing bits 7 and 3 of that port's status byte ($04F9 for port 1; $04FA for port 2).

Transmitting serial data

The serial firmware does not implement buffering for serial output. Instead it waits for two conditions to be true before transmitting a character:

- The ACIA's transmit register must be ready to accept a character. This is true if bit 4 of the ACIA's status register is 1.

- The device must signal that it is ready to accept data. This is true if bit 5 of the ACIA's status register is 0. Bit 5 is 0 if pin 5 of the port's connector is also 0.

When the serial firmware is active, a change of state on pin 5 of that port generates an interrupt. That interrupt is absorbed, but the data remains in bit 5 of the status register. Interrupts from the ACIA's transmit register are normally disabled.

A loophole in the firmware

So that programs can make use of interrupts on the ACIAs without affecting mouse interrupt handling, there is a tiny loophole purposely left in the built-in interrupt handler. If transmit interrupts are enabled on the ACIA—that is, if bits 3, 2, and 0 of the ACIA's command register have the values 0, 1, and 1, respectively—then control is passed to the user's interrupt handler if the interrupt is not intended for the mouse (movement, button, or VBL).

This means that you can write more sophisticated serial interrupt-handling routines than the limited firmware space could provide (such as printer spooling). The firmware will still set memory to its standard state, handle mouse interrupts, and restore memory after your routine is finished.

When you receive the interrupt, neither ACIA's status register has been read. You are fully responsible for checking for interrupts on both ACIAs, determining which of the four interrupt sources on each ACIA caused the interrupt, and how to handle them. Refer to the 6551 specification for more details. The built-in firmware itself is an excellent example of how interrupts on the ACIA can be handled.

Bypassing the interrupt firmware

The following sections give further details on using interrupts on the Apple IIc–family computers without using the built-in interrupt handler.

A method of handling mouse interrupts directly is described in Chapter 9.

Using mouse interrupts without the firmware

To use mouse interrupts without the firmware, as mentioned above, you must set your own interrupt vectors. If the $D000–$FFFF ROM is ever switched in, the built-in interrupt handler absorbs the mouse interrupts.

Tables 3-5 and 3-6 show how to activate and read mouse interrupts without using the firmware. Remember to disable interrupts (SEI) before enabling mouse interrupts, then turn them on again when done (CLI).

■ **Table 3-5** Activating mouse interrupts

To activate interrupts on	Enable IOU access	Select source	Enable source	Disable IOU access
Mouse X (rising edge)	STA $C079	STA $C05C	STA $C059	STA $C078
Mouse X (falling edge)	STA $C079	STA $C05D	STA $C059	STA $C078
Mouse Y (rising edge)	STA $C079	STA $C05E	STA $C059	STA $C078
Mouse Y (falling edge)	STA $C079	STA $C05F	STA $C059	STA $C078
VBL	STA $C079		STA $C05B	STA $C078

■ **Table 3-6** Reading mouse interrupts

To read interrupts from	Read direction (A.S.A.P)	Determine source	Handle it	Return
Mouse X	LDA $C066	LDA $C015 (bit 7=1 if true)	...	RTI
Mouse Y	LDA $C067	LDA $C017 (bit 7=1 if true)	...	RTI
VBL		LDA $C019 (bit 7=1 if true)	...	RTI

The mouse direction data read from $C066 and $C067 are guaranteed valid for at least 40 microseconds, and average duration is at least 200 microseconds, so you should read the direction as soon as possible.

Using ACIA interrupts without the firmware

To use ACIA interrupts without the firmware, you must set your own interrupt vectors. If the $D000–$FFFF ROM is ever switched in, the built-in interrupt handler handles the interrupt as determined by certain mode bytes.

When writing your serial interrupt handler, refer to Figures 11-37 through 11-39 and to the Synertek 6551 ACIA specification. As shown in Chapter 11, the ACIAs have the following connections:

Port 1 DSR line (pin 17) connected to the EXTINT line on the external disk port; connected to +5 volts through a 3.35 ohm resistor on the Apple IIc Plus. DCD line (pin 16) connected to pin 5 of port 1 connector.

Port 2 DSR line connected to keyboard strobe; goes high when a key is pressed. DCD line connected to pin 5 of port 2 connector.

The ACIA registers have the following addresses:

Port 1		**Port 2**	
Data register	$C098	Data register	$C0A8
Status register	$C099	Status register	$C0A9
Command register	$C09A	Command register	$C0AA
Control register	$C09B	Control register	$C0AB

Chapter 4 **Introduction to Apple IIc I/O**

This chapter is an introduction to the built-in I/O capabilities of the Apple IIc and Apple IIc Plus computers. This chapter outlines

- standard I/O links and their functions
- I/O firmware protocols
- dedicated memory storage locations
- direct I/O ■

The standard I/O links

You can use some of the routines in the Apple IIc family's firmware for your own programs. This can save you both program space and the time and effort of writing all your own I/O routines.

There are two levels of firmware subroutines used for I/O; one level is the I/O subroutines, the other level is the standard input/output routines. When your program makes a call to an I/O subroutine, the called routine checks a standard location in memory for the address of the input or output routine to which it should pass control. The address to which the I/O subroutine passes control is called a **vector,** and the standard locations in memory that contain the vectors used by I/O subroutines are called the **I/O links.**

To use an I/O subroutine, your program must perform a JSR instruction to the routine's address (its entry point). The called routine then performs an indirect jump instruction to the address pointed to by the standard I/O link and begins executing. When the routine is finished, it executes an RTS instruction and control is returned to your program at the first instruction following the original JSR instruction.

◆ *Note:* JSR means *Jump to SubRoutine.* It is a machine-language statement that causes control to be turned over to the routine loaded at a specified address (called the *jump address*).

An **indirect jump** jumps to a vector pointing it to the start of the routine.

RTS means *Return To Statement.* It is a machine-language statement that causes the routine to end and control to be passed back to the application program or operating system that initiated the call at the statement immediatly following the call.

In an Apple IIc or Apple IIc Plus running without an operating system, each I/O link normally contains the address of the standard input or output routine. The reason to use this indirect approach instead of calling the standard input/output routines directly is that an operating system might substitute its own input or output routines for the standard ones in firmware. For example, an operating system might redirect output to go to a file rather than to the screen. If your program calls the standard input/output routines in ROM rather than the I/O subroutines, it bypasses the operating system and may not work properly with new versions of the operating system or firmware.

◆ *Note:* Not all operating systems use the standard I/O links. For example, Apple Pascal does not use them.

The standard I/O links are at locations \$36 and \$38. The link at location \$36 is called *CSW* (character output switch). Individually, location \$36 is called *CSWL* (CSW low) and location \$37 is called *CSWH* (CSW high). The CSW link holds the starting address of the subroutine the computer is currently using for single-character output.

The link at location \$38 is called *KSW* (keyboard input switch). Individually, location \$38 is called *KSWL* (KSW low) and location \$39 is called *KSWH* (KSW high). This link holds the starting address of the routine currently being used for single-character input.

When you issue either a PR#*n* from BASIC or an *n* Control-P from the System Monitor, the Apple IIc or Apple IIc Plus changes the value in the CSW link to the first address in the ROM space allocated to port *n*. That address is \$Cn00. The next time a character is output using the standard output routine, control is transferred to the firmware starting at \$Cn00. When it has finished, the firmware executes an RTS instruction to return control to the calling program. Sometimes the firmware at \$Cn00 changes both input and output links; for example, when you issue a PR#3 to invoke the enhanced video firmware, the routine that starts at \$C300 changes both the input and output links to point to enhanced video firmware routines. Enhanced video firmware is described in the section "Enhanced Video Firmware" in Chapter 6.

When you issue an IN#*n* command from BASIC or an *n* Control-K from the Monitor, the Apple IIc or Apple IIc Plus changes the value in the KSW link to the first address in the ROM space allocated to port *n*; that is, to \$Cn00. The IN#*n* command and *n* Control-K command work in a similar fashion to the output-link commands: the next call for character input is transferred to the firmware starting at \$Cn00.

Notice that not all ports can be used for I/O. The firmware starting at \$C600, the 5.25-inch disk drive port, for example, causes the computer to attempt to restart from a disk in the first 5.25-inch disk drive.

When a disk operating system (DOS or ProDOS) is running, one or both of the standard I/O links hold addresses of the disk operating system's input and output routines. The operating system has internal locations that hold the addresses of the currently active character input and output routines.

▲ **Warning** If a program that is running with DOS or ProDOS changes the values in the standard I/O links, either directly or via IN# and PR# commands, the operating system may be disconnected from the system. To avoid this problem when programming in BASIC, you should always issue an empty PRINT statement followed by a PRINT statement containing Control-D and the IN# or PR# command. Applesoft passes on to the operating system any command following a carriage return–Control-D. ▲

△ DOS 3.3 After changing either CSW or KSW, your assembly-language programs running under DOS 3.3 should call the subroutine at location $03EA. This subroutine transfers the link address to a location inside the operating system and then restores the operating system link address in the standard link location. △

The I/O links contain the addresses of the KeyIn and COut1 routines if the enhanced video firmware is off (when the display shows a flashing checkerboard cursor), and of the C3KeyIn and C3COut1 routines if the enhanced video firmware is on (when the display shows an inverse solid cursor). All of these I/O routines are described in this chapter. The enhanced video firmware is described in Chapter 6.

Standard input routines

The Apple IIc family's firmware includes two different subroutines that your program can call in order to read data from the keyboard: *RdKey* (read key) and *GetLn* (get line).

RdKey calls the current character input routine (that is, the one whose address is stored at KSW). This routine is normally KeyIn or C3KeyIn, either of which accepts one character from the keyboard. GetLn repeatedly calls the current character input routine to allow your program to get a *sequence* of characters terminated with a carriage return. Thus GetLn allows line-oriented input using the current input routine. GetLn also provides some on-screen editing features, as described in the section "Editing With GetLn," later in this chapter.

RdKey subroutine

A program can get a character from the keyboard by making a subroutine call to RdKey at memory location $FD0C. RdKey passes control via the input link KSW to the current input routine.

RdKey displays a cursor at the current cursor position, which is immediately to the right of whatever character you last sent to the display (normally by using the COut routine, described later in this chapter).

GetLn subroutine

Programs often need strings of characters as input. While you could call RdKey repeatedly to get several characters from the keyboard, there is an easier way to do it. The routine that you can use in this case is named *GetLn,* and it starts at location $FD6A. Using repeated calls to RdKey, GetLn obtains characters from the current input routine and puts them into the input buffer located in the memory page from $0200 to $02FF. GetLn also provides some basic on-screen editing and control features.

The first thing GetLn does when you call it is to display a prompt. The prompt indicates to the user that the program is waiting for input. Different programs use different prompt characters, helping to remind the user which program is requesting the input. Table 4-1 shows the prompt characters used by standard programs on the Apple IIc family.

GetLn uses the character stored at memory location $33 as the prompt character. In an assembly-language program, you can change the prompt to any character you wish. In BASIC, changing the prompt character has no effect because both BASIC interpreters and the Monitor restore it each time they request input from the user.

◆ *Note:* Applesoft uses GetLn1 ($FD6F) when a program is executing. GetLn1 does not print a prompt.

■ **Table 4-1** Prompt characters

Prompt character	Program requesting input
?	User's BASIC program (INPUT statement)
]	Applesoft BASIC (Appendix C)
>	Integer BASIC (Appendix C)
*	Firmware Monitor (Chapter 10)

As the user types each character, GetLn sends the character to the current output routine, which displays it at the current cursor position and then advances the cursor to indicate the next character position. Control characters echoed by GetLn are not executed.

GetLn stores the characters in its buffer, which starts at memory location $0200. GetLn uses the X register to index the buffer. GetLn continues to accept and display characters until the user presses Return (the user can also press Control-X to cancel the line). Then it clears the remainder of the line the cursor is on, stores the carriage-return code to mark the end of the buffer, places the cursor at the beginning of the next line, and returns to the routine that called it.

The maximum line length that GetLn can handle is 255 characters. If the user types more than this, GetLn sends a backslash (\) and a carriage return to the display, cancels the line it has accepted so far, and starts over. To warn the user that the line is getting full, GetLn sounds a bell (tone) at every keypress after the 248th.

◆ *Note:* The Applesoft interpreter accepts only 239 characters.

Escape sequences with GetLn

GetLn has many special functions that you invoke by typing escape sequences on the keyboard. An escape sequence is sent by pressing Esc (Escape), releasing it, and then pressing some other key, as shown in Table 4-2.

△ **Important** Be sure to release Esc right away. If you hold it too long, the auto-repeat mechanism begins, which may cancel the escape sequence. △

■ **Table 4-2** Escape sequences with GetLn

Escape code	Function
Esc	Clears the window and homes the cursor (places it in the upper-left corner of the screen); exits from escape mode
Esc A or Esc a	Moves the cursor right one line; exits from escape mode
Esc B or Esc b	Moves the cursor left one line; exits from escape mode
Esc C or Esc c	Moves the cursor down one line; exits from escape mode
Esc D or Esc d	Moves the cursor up one line; exits from escape mode
Esc E or Esc e	Clears to the end of the line; exits from escape mode
Esc F or Esc f	Clears to the bottom of the window; exits from escape mode
Esc I or Esc i *or* Esc Up Arrow	Moves the cursor up one line; remains in escape mode
Esc J or Esc j *or* Esc Left Arrow	Moves the cursor left one space; remains in escape mode*
Esc K or Esc k *or* Esc Right Arrow	Moves the cursor right one space; remains in escape mode*
Esc M or Esc m *or* Esc Down Arrow	Moves the cursor down one line; remains in escape mode*
Esc 4	Switches to 40-column mode; sets links to C3KeyIn and C3COut1; restores normal window size (Table 4-5); exits from escape mode†
Esc 8	Switches to 80-column mode; sets links to C3KeyIn and C3COut1; restores normal window size (Table 4-5); exits from escape mode†
Esc Control-D	Disables control characters; only carriage return, line feed, bell, and backspace have an effect when printed
Esc Control-E	Reactivates control characters
Esc Control-Q	Deactivates the enhanced video firmware; sets links to KeyIn and COut1; restores normal window size (Table 4-5); exits from escape mode†

* Cursor-control key: see text.

† This code functions only when the enhanced video firmware is active.

In escape mode, you can keep using the arrow keys and the cursor movement keys I, J, K, and M without pressing Esc again. This enables you to perform repeated cursor moves by holding down the appropriate key.

When GetLn is in escape mode, it displays an inverse plus sign as the cursor. You leave escape mode by typing any key other than a cursor movement key.

◆ *Note:* The escape sequences that use the arrow keys are the standard cursor movement keys on the Apple IIc family. The escape sequences that use I, J, K, and M are the standard cursor movement keys on the Apple II and Apple II Plus, and are present on the Apple IIc family for compatibility.

Escape sequences can be used in the middle of an input line to change the appearance of the screen. They have no effect on the input line.

Editing with GetLn

The GetLn subroutine provides the standard on-screen editing features used by the BASIC interpreters and the Monitor. Any program that uses GetLn for reading the keyboard has these features.

Cancel line

Any time you are typing a line, pressing Control-X causes GetLn to cancel the line. GetLn displays a backslash (\) and issues a carriage return, then displays the prompt and waits for you to type a new line. GetLn takes the same action when you type more than 255 characters, as described earlier.

Backspace

When you press Left Arrow (or Control-H), GetLn moves its buffer pointer back one space, effectively deleting the last character in its buffer. It also sends a backspace character to routine COut, which moves the cursor back one space. If you type another character now, it replaces the character you backspaced over, both on the display and in the line buffer.

Each time you press Left Arrow, it moves the cursor left and deletes another character, until you are back at the beginning of the line. If you then press Left Arrow one more time, you have effectively canceled the line, and GetLn issues a carriage return and displays the prompt. The cursor moves even if the deleted character is an invisible control character. Thus it is possible for screen alignment and buffer alignment to be different.

Retype

Right Arrow (or Control-U) has a function that is complementary to the backspace function. When you press Right Arrow, GetLn picks up the character under the cursor just as if it had been typed on the keyboard. You can use this procedure to pick up characters that you just deleted by backspacing across them. You can use the backspace and retype functions with the cursor-motion functions to edit data on the display.

The KeyIn and C3KeyIn input routines

KeyIn is the standard input routine that is used when the enhanced video firmware (described in Chapter 6) is not in use. C3KeyIn is the standard input routine that is used when the enhanced video firmware is active. When called (usually by RdKey or by GetLn), either routine displays a cursor, waits until the user presses a key, then inserts the ASCII code of the key just pressed in the A register (accumulator) and returns to the calling program.

KeyIn displays a cursor by alternately storing a checkerboard block in the memory location used for the cursor character, storing the original character, then storing the checkerboard again. C3KeyIn places a block cursor on the screen by inverting (swapping black for white) the character at the cursor position.

KeyIn and C3KeyIn also generate a random number. While it is waiting for the user to press a key, the input routine repeatedly increments the 16-bit number in memory locations $4E and $4F. This number keeps increasing from 0 to $FFFF (65535), then starts over again at 0. The value of this number changes so rapidly that it is very difficult to predict what it will be after a key is pressed. A program that reads from the keyboard can use this value as a random number or as a seed for a pseudo-random number routine.

Standard output routines

The output subroutine provided for your use is named *COut* (character output). COut calls the current character output routine (that is, the one whose address is stored at CSW). This routine is normally COut1 or C3COut1, either of which sends one character to the display, advances the cursor position, and scrolls the display when necessary. COut1 and C3COut1 restrict their use of the display to an active area called the *text window*, described later in this chapter.

COut subroutine

When your program places a character in the A register and makes a subroutine call to COut at memory location $FDED, COut passes control via the output link CSW to the current output routine. If the output routine is COut1 or C3COut1, and the A register contains an uppercase or lowercase letter, a number, or a special character, the routine displays the character on the screen. If the A register contains a control character, COut1 or C3COut1 either performs one of the special functions described below or ignores the character.

Each time you send a displayable character to COut1 or C3COut1, it displays the character at the current cursor position, replacing whatever was there, and then advances the cursor position one space to the right. If the cursor position is already at the right edge of the window, COut1 or C3COut1 moves it to the leftmost position on the next line down. If this would move the cursor position past the end of the last line in the window, COut1 or C3COut1 scrolls the display up one line and sets the cursor position at the left end of the new bottom line.

The cursor position is controlled by the values in memory locations $24 and $25. These locations are named *CH*, for cursor horizontal, and *CV*, for cursor vertical. COut1 and C3COut1 do not display a cursor, but the input routines (such as KeyIn or C3KeyIn) do, and they use this cursor position. However, changing CV directly does not change the cursor's vertical position until the next carriage return or until reaching the end of the current line causes a call to VTab (for setting the base address within windows). If some other routine displays a cursor, it does not necessarily put it in the cursor position used by COut1 or C3COut1.

△ Important — When the enhanced video firmware is active, the value of CH is kept at 0 and the true horizontal position is stored at $057B. When the enhance video firmware is active, use $057B instead of CH. △

Control characters with COut1

COut1 does not display control characters. Instead, the control characters listed in Table 4-3 are used to initiate some action by the firmware. Other control characters are ignored. Most of the functions listed here can also be invoked from the keyboard, either by typing the control character listed or by using the one of the escape sequences shown in Table 4-2. The stop-list function, described separately, can only be invoked from the keyboard.

■ **Table 4-3** Control characters with COut1

Control character	ASCII name	Apple IIc name	Action taken by COut1
Control-G	BEL	Bell	Produces a 1000 Hz tone for 0.1 second
Control-H	BS	Backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above
Control-J	LF	Line feed	Moves cursor position down to next line in window; scrolls if needed
Control-M	CR	Return	Moves cursor position to left end of next line in window; scrolls if needed
Control-S	DC3	Stop-list	Stops listing characters on the display until another key is pressed†

† Only works from the keyboard.

Control characters with C3COut1

When the enhanced video firmware is active, COut calls C3COut1 instead of COut1 for character output. C3COut1 does not display control characters, but you can use certain control characters to control some functions. All other control characters are ignored.

The control characters listed in Table 4-4 are used to initiate some action by the firmware. Except for the stop-list function (Control-S) you can send control characters to C3COut1 either from a program or from the keyboard. The stop-list function can only be invoked from the keyboard. Most of the functions listed here can also be performed by using one of the escape sequences shown in Table 4-2.

■ **Table 4-4** Control characters with C3COut1

Control character	ASCII name	Apple IIc name	Action taken by C3COut1
Control-G	BEL	Bell	Produces a 1000-Hz tone for 0.1 second
Control-H	BS	Backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above
Control-J	LF	Line feed	Moves cursor position down to next line in window; scrolls if needed
Control-K	VT	Clear EOS	Clears from cursor position to the end of the screen*
Control-L	FF	Home and clear	Moves cursor position to upper-left corner of window and clears window*
Control-M	CR	Return	Moves cursor position to left end of next line in window; scrolls if needed
Control-N	SO	Normal	Sets display format normal*
Control-O	SI	Inverse	Sets display format inverse*
Control-Q	DC1	40-column	Sets display to 40-column*
Control-R	DC2	80-column	Sets display to 80-column*
Control-S	DC3	Stop-list	Stops listing characters on the display until another key is pressed†
Control-U	NAK	Quit	Turns off enhanced video firmware*
Control-V	SYN	Scroll	Scrolls the display down one line, leaving the cursor in the current position*
Control-W	ETB	Scroll-up	Scrolls the display up one line, leaving the cursor in the current position*
Control-X	CAN	Disable MouseText	Disables MouseText character display; uses inverse uppercase*
Control-Y	EM	Home	Moves cursor position to upper-left corner of window (but doesn't clear)*
Control-Z	SUB	Clear line	Clears the line the cursor position is on*
Control-[	ESC	Enable MouseText	Maps inverse uppercase characters to MouseText characters*
Control-\	FS	Fwd. space	Moves cursor position one space to the right; from right edge of window, moves it to left end of line below*
Control-]	GS	Clear EOL	Clears from the current cursor position to the end of the line (that is, to the right edge of the window)*
Control-_	US	Up	Moves cursor up a line, no scroll*

* Doesn't work from the keyboard.
† Works only from the keyboard.

The stop-list feature

You can stop the computer from updating its display (if it is using either COut1 or C3COut1) by pressing Control-S. Whenever COut1 or C3COut1 gets a carriage return from the program, it checks the keyboard for a Control-S. If a Control-S has been pressed, COut1 or C3COut1 stops and waits for another key to be pressed before resuming. The character code of the key that is pressed is ignored unless it is Control-C, which is passed to the program. This last feature lets you exit BASIC programs from stop-list mode.

The text window

The active portion of the display is called the *text window*. After you start up the computer or perform a reset, the entire display is the text window. COut1 or C3COut1 puts characters only into the window; when it reaches the end of the last line in the window, it scrolls only the contents of the window.

You can restrict video activity to any rectangular portion of the display by changing the current text window. Your programs can thus control the placement of text in the display and protect other portions of the screen from being written over by new text. To do this, store the appropriate values into four locations in memory to set the top, bottom, left margin, and width of the text window. The following memory locations control the text window:

- The left margin is stored in memory location $20. This number is normally 0, the number of the leftmost column in the display. In a 40-column display, the maximum value for this number is 39 (hexadecimal $27); in an 80-column display, the maximum value is 79 (hexadecimal $4F).
- The width of the text window is stored in memory location $21. For a 40-column display, this value is normally 40 (hexadecimal $28); for an 80-column display, it is normally 80 (hexadecimal $50).
- The position of the top line of the text window is stored in memory location $22. This is normally 0, the topmost line in the display. Its maximum value is 23 (hexadecimal $17).
- The position of the bottom line of the screen plus 1 is stored in memory location $23. It is normally 24 (hexadecimal $18) for the bottom line of the display. Its minimum value is 1.

△ Important Pascal does not use this method of supporting window widths. △

▲ **Warning** Be careful not to let the sum of the window width and the leftmost position in the window exceed the width of the display you are using (40 or 80 columns). If this happens, COut1 or C3COut1 may put characters into memory locations outside the display page, possibly destroying programs or data. ▲

Table 4-5 summarizes the memory locations and the possible values for the text window parameters.

■ **Table 4-5** Text window memory locations

Window	Location		Minimum value		Normal values 40-col.		Normal values 80-col.		Maximum values 40-col.		Maximum values 80-col.	
parameter	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
Left edge	32	$20	00	$00	00	$00	00	$00	39	$27	79	$4F
Width	33	$21	00	$00	40	$28	80	$50	40	$28	80	$50
Top edge	34	$22	00	$00	00	$00	00	$00	23	$17	23	$17
Bottom edge	35	$23	01	$01	24	$18	24	$18	24	$18	24	$18

Normal, inverse, and flashing text

The way that the Apple IIc family displays characters is affected by two things: the value that is stored in the inverse flag (zero-page location $32), and whether the enhanced video firmware (described in Chapter 6) is off or on. The inverse flag's influence is discussed in the next two subsections.

If the enhanced video firmware is off, the Apple IIc family displays what is called the *primary character set;* if the enhanced video firmware is on, the Apple IIc family displays what is called the *alternate character set.*

The primary character set includes normal (light on dark), inverse (dark on light), and flashing (alternating normal and inverse) characters. Lowercase inverse characters are not included in the primary character set. The alternate character set includes normal and inverse characters (including lowercase inverse), and a set of graphic characters called *MouseText*. Flashing characters are not included in the alternate character set. These character sets are described in more detail in the section "Text Character Sets," in Chapter 6.

If you want your program to display a character, it should first load the character to be displayed in the A register, and then call the character-output subroutine COut. For example, to display the character corresponding to $C8, you can use something like this:

```
LDA #$C8
JSR COut
```

Primary-character-set display

The primary character set is displayed by COut1, which operates only when the enhanced video firmware is off. The primary character set includes text in normal, inverse, or flashing format, but not inverse or flashing lowercase text.

If the value of the character sent to COut1 is greater than or equal to $A0, COut1 performs a logical AND operation on the value of the character with the value of the inverse flag (at location $32), and the result is displayed. (If you're curious about which ASCII character is being sent, subtract $80 from the value being sent to COut1.) You can use the following inverse flag values:

- $FF (decimal 255) produces the normal character format.
- $3F (decimal 63) produces the inverse character format.
- $7F (decimal 127) produces the flashing character format.

△ Important — To avoid unusual character display results, use only the three values $3F, $7F, and $FF. △

COut1 interprets character values from $80 through $9F as control characters and tries to execute them.

Character values from $00 through $7F are all interpreted as display characters, not control characters.

Alternate-character-set display

The alternate character set includes normal and inverse format characters and the MouseText graphic characters. You should use C3COut1, the standard output link when the enhanced video firmware is *active*, to display the alternate character set. Here are the rules for using the alternate character set:

- Control characters are not displayed. Characters sent to C3COut1 are interpreted as control characters if they are in the range $00 through $1F or $80 through $9F.
- Characters in the range $20 through $7F and $A0 through $FF are displayed.
- If inverse flag (location $32) bit 7 is 1, the character is normal.
- If inverse flag bit 7 is 0, the character is inverse.
- If MouseText is off, characters $40 through $5F are remapped to the range $00 through $1F and are displayed as uppercase inverse characters.
- If MouseText is on, character values $40 through $5F are left unchanged, and the characters are displayed as MouseText.

See the section "MouseText," in Chapter 6, for more information about MouseText.

Port I/O

The Apple IIc and Apple IIc Plus are members of the Apple II family of computers; however, unlike the Apple II, II Plus, IIe, and IIGS, the Apple IIc and Apple IIc Plus do not have peripheral connector slots. In place of these, they have **ports**—the built-in firmware and hardware equivalent of the expansion cards installed in slots in the other Apple II computers.

Standard port entry points

To maintain compatibility with existing software and its protocols, each port's I/O firmware has the same standard entry point ($Cn00) as its equivalent slot in another Apple II would have. Tables 4-6 and 4-7 show these equivalents for the various versions of the Apple IIc, as well as listing the chapter where each port is described. Connectors and hardware descriptions of ports are given in Chapter 11.

Executing a PR#*n* or IN#*n* command from Applesoft changes the standard output or input links, respectively, so that the next output or input through the standard I/O links causes a jump to the firmware entry point at address $Cn00. See the earlier section "The Standard I/O Links," earlier in this chapter, for more information on the standard I/O links.

■ **Table 4-6** Original and UniDisk 3.5 Apple IIc port characteristics

Port	Entry point	Port connector	Use	Chapter
1	$C100	Serial port 1	Printers	7
2	$C200	Serial port 2	Communication	7
3	$C300	Video connectors	Enhanced video firmware	6
4	$C400	Mouse	Mouse	9
5	$C500	Intelligent disk port devices	External 3.5-inch drives*	8
6	$C600	Disk drives	Built-in and external drives	8
7	$C700	No device	Reserved	

* UniDisk 3.5 Apple IIc only

■ **Table 4-7** Memory expansion Apple IIc and Apple IIc Plus port characteristics

Port	Entry point	Port connector	Use	Chapter
1	$C100	Serial port 1	Printers	7
2	$C200	Serial port 2*	Communication	7
3	$C300	Video connectors	Enhanced video firmware	6
4	$C400	Memory expansion card	Memory expansion card	8
5	$C500	3.5 inch disk drives	Built-in and external 3.5-inch drives	8
6	$C600	5.25 inch disk drives	Built-in and external 5.25-inch drives	8
7	$C700	Mouse	Mouse	9

* Also supports second and third external 3.5-inch drives.

Firmware protocol

The Apple IIc family supports a standard firmware protocol that, in addition to the standard link address, provides a table of device identification and entry points to standard and optional firmware subroutines. The protocol is equivalent to the Pascal 1.1 firmware protocol in use on other Apple II computers, and is outlined in Table 4-8. In Table 4-8, *n* is the port number.

■ **Table 4-8** Firmware protocol locations

Address	Value	Description
$Cn05	$38	Pascal firmware card/port identifier.
$Cn07	$18	Pascal firmware card/port identifier.
$Cn0B	$01	Generic signature byte of a firmware card/port.
$Cn0C	$ci	Device signature byte: *i* is an identifier (not necessarily unique). c = device class (not all used in the Apple IIc family): $00 reserved $01 printer $02 hand control or other X-Y device $03 serial or parallel I/O card/port $04 modem $05 sound or speech device $06 clock $07 mass-storage device $08 80-column card/port $09 network or bus interface $0A special purpose (none of the above) $0B–0F reserved
$Cn0D	ii	$Cnii is the initialization entry address (PInit).
$Cn0E	rr	$Cnrr is the read routine entry address (PRead) (returns character read in A register).
$Cn0F	ww	$Cnww is the write routine entry address (PWrite) (enters with character to write in A register).
$Cn10	ss	$Cnss is the status routine entry address (PStatus) (enters with request code in A register: 0 to ask "Are you ready to accept output?" or 1 to ask "Do you have input ready?").
$Cn11	$00	If additional address bytes follow; nonzero if not.

Each table begins with identification bytes ($Cn05 through $Cn0C). Then, starting with address $Cn0D, each byte in the table represents the low-order byte of the entry-point address of a firmware routine. The high-order byte of each address is always $Cn, where *n* is the port number. Your program uses these byte values to construct its own jump table for subroutine calls to the ports.

All port routines require, on entry, that the X register contain $Cn and that the Y register contain $n0.

All routines, on exit, return an error code in the X register (0 means no error occurred; 3 means the request was invalid). The carry bit in the program status register usually contains a reply to a request code (0 means *no;* 1 means *yes*).

All the ports in the Apple IIc family except the disk port conform to this protocol. The disk port is described in Chapter 8.

Port I/O space

By a convention used in other Apple II–series machines, each port or slot has exclusive use of 16 memory locations set aside for data input and output. The addresses of these locations are of the form $C080 + $n0, where *n* is the port or slot number. Table 4-9 lists the port I/O space used in the Apple IIc family. See the section "The Hardware Page," in Appendix B, for the assignments of specific addresses.

■ **Table 4-9** Port I/O locations

Port	Locations
1	$C090–$C09F
2	$C0A0–$C0AF
5	$C0D0–$C0DF
6	$C0E0–$C0EF

Port ROM space

In the Apple II and IIe, one 256-byte page of address space is allocated to each slot. This space is used for read-only memory (ROM or PROM on the interface card) with driver programs that control the operation of input/output devices, as outlined in Table 4-8. In the Apple IIc family, this space is dedicated to port firmware. However, I/O ROM space in the Apple IIc family is used as efficiently as possible, and there is *not* a strict correspondence between firmware for port *n* and the $Cn00 space, except that the entry point for port *n* is always at $Cn00.

Expansion ROM space

The 2 KB memory space from $C800 to $CFFF in the Apple IIc–family main ROM contains the enhanced video firmware, port 3, and subroutines used to transfer data between locations in memory. The Apple IIc family, unlike the Apple II, Apple II Plus, or Apple IIe, always has this space switched in; that is, when the system switches between ROM and RAM, the $C800 through $CFFF address space remains assigned to ROM.

Port screen hole RAM space

There are 128 bytes of memory (64 in main memory, 64 in auxiliary memory) allocated to the ports, 8 bytes per port, as shown in Table 4-10. These bytes are reserved for use by the system, except as described in Chapters 5 through 9.

■ **Table 4-10** Port screen hole memory locations

	Ports						
Base address	**1**	**2**	**3**	**4**	**5**	**6**	**7**
$0478	$0479	$047A	$047B	$047C	$047D	$047E	$047F
$04F8	$04F9	$04FA	$04FB	$04FC	$04FD	$04FE	$04FF
$0578	$0579	$057A	$057B	$057C	$057D	$057E	$057F
$05F8	$05F9	$05FA	$05FB	$05FC	$05FD	$05FE	$05FF
$0678	$0679	$067A	$067B	$067C	$067D	$067E	$067F
$06F8	$06F9	$06FA	$06FB	$06FC	$06FD	$06FE	$06FF
$0778	$0779	$077A	$077B	$077C	$077D	$077E	$077F
$07F8	$07F9	$07FA	$07FB	$07FC	$07FD	$07FE	$07FF

These addresses are unused bytes in the RAM reserved for text and Lo-Res graphics displays, and hence they are called *screen holes*. These particular locations are not displayed on the screen and their contents are not changed by the built-in output routines. In other words, they are used by the output routines, but they are not part of the video display.

▲ **Warning** All the screen holes in auxiliary memory, and many of them in main memory, are reserved for special use by Apple IIc-family firmware—for example, to store initialization information. Do not use any locations marked *reserved* in this manual. ▲

The way that port firmware uses these RAM locations and their addresses is covered in Chapters 5 through 9.

Interrupts

Interrupts are a way to more efficiently use the hardware in a computer. Interrupt support built into the Apple IIc family's firmware is described briefly in this section. For a complete description of interrupts in the Apple IIc-family computers, see Chapter 3.

When the IRQ line on the 65C02 microprocessor is activated, the 65C02 transfers program control through the vector in locations $FFFE through $FFFF of ROM or whichever bank of RAM is switched in. If ROM is switched in, this vector is the address of the Monitor's interrupt handler, which determines whether the request is due to an interrupt that should be handled internally. If so, the Monitor handles it and then returns control to the interrupted program.

If the interrupt is due to a BRK ($00) instruction, control is transferred through the BRK vector ($03F0–$03F1). Otherwise, control is transferred through the IRQ vector ($03FE–$03FF).

Chapter 5 **Keyboard and Speaker**

This chapter describes how your program can use the Apple IIc keyboard and the speaker. Notice that the keyboard used in the Apple IIc Plus computer differs from older Apple IIc keyboards only in the layout and labeling of some of the keys; the Apple IIc Plus keyboard functions in exactly the same way as older keyboards.

See Chapter 11 for physical descriptions of the keyboards used in the Apple IIc and Apple IIc Plus computers. ■

Keyboard input

Table 5-1 describes the characteristics of the keyboard that relate to programming. You won't have to write routines to read the keyboard from your assembly-language programs since the Apple IIc family System Monitor provides keyboard support through the three standard input routines described in Chapter 4—RdKey, KeyIn, and GetLn. You *can* do all your keyboard handling directly in your programs if you want to, but it's nice to know that you're not forced to.

Reading the keyboard

The keyboard encoder and ROM (see Chapter 11) can generate all 128 ASCII codes, so all the special character codes in the ASCII character set are available from the keyboard. Your machine-language programs can call RdKey to get characters from the keyboard. RdKey reads characters a byte at a time from the keyboard data location ($C000).

Here is how your programs should go about reading the keyboard:

1. Test bit 7 of address $C000 to see if a key has been pressed. Bit 7 is the keyboard strobe bit.
2. When bit 7 goes to a 1, you know that the low-order seven bits of $C000 are a valid character.
3. Clear the keyboard strobe (bit 7) at $C000 by reading or writing *anything* to address $C010.

$C010 has another function besides clearing the keyboard strobe: its high bit is a 1 while a key is pressed (except Command (Open Apple), Option (Solid Apple), Control, Shift, Caps Lock, and Reset). Bit 7 at this location is therefore called *any-key-down*. You could use this to let a program do something useful other than just waiting for the next key to be pressed. (People are generally a *lot* slower than the Apple IIc–family computers.) Check $C010 occasionally to see if something should be done.

△ Important — If your program needs to read both the any-key-down flag and the keyboard strobe, it must read the strobe bit first. Any time you read the any-key-down bit at $C010, you also clear the keyboard strobe bit at $C000. △

■ **Table 5-1** Keyboard input characteristics

Port number	None
Commands	Keyboard is always on, in the sense that any keypress generates a KSTRB signal.
Initial characteristics	Reset routine clears the keyboard strobe and sets the keyboard as the standard input device (that is, sets KSW to point to RdKey).
Hardware locations	
$C000	Keyboard data and strobe
$C010	Any-key-down flag and clear-strobe switch
$C060	40/80-column switch status on bit 7; 1 = 40-column display (switch down)*
$C061	Command key status on bit 7; 1 = pressed (also game input switch 0 and first mouse button status†)
$C062	Option key status on bit 7; 1 = pressed

Monitor firmware routines

Location	Name	Description
$FD6A	GetLn	Gets an input line with prompt
$FD67	GetLnZ	Gets an input line with preceding carriage return
$FD6F	GetLn1	Gets an input line, but with no preceding prompt
$FD1B	KeyIn	The keyboard input subroutine
$FD35	RdChar	Gets an input character or escape code
$FD0C	RdKey	The standard character input subroutine

Use of other pages

Page 2 The standard character string input buffer (see GetLn description)

* The Apple IIc Plus does not have a 40/80 column switch.

† Game input soft switches are listed and described in the section "Game Input," in Chapter 9.

After your program has cleared the keyboard strobe, the strobe remains low until another key is pressed.

Table 5-2 shows the ASCII codes generated by all the keys on the Apple IIc keyboard. The Apple IIc Plus keyboard generates the same key codes as the older Apple IIc keyboards.

◆ *Note:* If the strobe bit is set, the character values that your program sees will be equal to the values given in Table 5-2 plus $80.

■ **Table 5-2** Keys and ASCII codes

	Key alone		+ Control		+ Shift		+ Both	
Key	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**
Delete	7F	DEL	7F	DEL	7F	DEL	7F	DEL
Left Arrow	08	BS	08	BS	08	BS	08	BS
Tab	09	HT	09	HT	09	HT	09	HT
Down Arrow	0A	LF	0A	LF	0A	LF	0A	LF
Up Arrow	0B	VT	0B	VT	0B	VT	0B	VT
Return	0D	CR	0D	CR	0D	CR	0D	CR
Right Arrow	15	NAK	15	NAK	15	NAK	15	NAK
Esc	1B	ESC	1B	ESC	1B	ESC	1B	ESC
Space	20	SP	20	SP	20	SP	20	SP
' "	27	'	27	'	22	"	22	"
, <	2C	,	2C	,	3C	<	3C	<
- _	2D	-	1F	US	5F	_	1F	US
. >	2E	.	2E	.	3E	>	3E	>
/ ?	2F	/	2F	/	3F	?	3F	?
0)	30	0	30	0	29	)	29	)
1 !	31	1	31	1	21	!	21	!
2 @	32	2	00	NUL	40	@	00	NUL
3 #	33	3	33	3	23	#	23	#
4 $	34	4	34	4	24	$	24	$
5 %	35	5	35	5	25	%	25	%
6 ^	36	6	1E	RS	5E	^	1E	RS
7 &	37	7	37	7	26	&	26	&
8 *	38	8	38	8	2A	*	2A	*
9 (	39	9	39	9	28	(	28	(
; :	3B	;	3B	;	3A	:	3A	:
= +	3D	=	3D	=	2B	+	2B	+
[{	5B	[	1B	ESC	7B	{	1B	ESC
\ |	5C	\	1C	FS	7C	|	1C	FS
] }	5D	]	1D	GS	7D	}	1D	GS
! ~	60	!	60	!	7E	~	7E	~

■ **Table 5-2** Keys and ASCII codes (continued)

	Key alone		+ Control		+ Shift		+ Both	
Key	Code	Char	Code	Char	Code	Char	Code	Char
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB
X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

There are three keys that do not generate ASCII codes themselves, but alter the characters produced by other keys. These modifier keys are Control, Shift, and Caps Lock.

You can also use the Command (Open Apple) and Option (Solid Apple) keys as character modifier keys while handling keyboard input. You can read Command at $C061 and Option at $C062.

Another key that doesn't generate a code is Reset, located at the upper-left corner of the keyboard; it is connected directly to the microprocessor RESET line. Pressing Control-Reset causes a warm reset. Pressing Command -Control-Reset causes a forced cold reset. The reset procedures are described in Chapter 4.

Monitor firmware support for keyboard input

Chapter 4 describes the three standard Monitor input routines serving the keyboard: GetLn, RdKey, and KeyIn. This section discusses the three other Monitor routines that serve the keyboard.

GetLnZ

GetLnZ (at address $FD67) is an alternate entry point for GetLn that first sends a carriage return to the standard output, then continues into GetLn.

GetLn1

GetLn1 (at address $FD6F) is an alternate entry point for GetLn that does not issue a prompt before it accepts the input line. However, if the user cancels the input line with too many backspaces or with Control-X, then GetLn1 issues the prompt stored at location $33 when it gets another line.

RdChar

RdChar (at address $FD35) is a subroutine that gets characters from the standard input subroutine, and also interprets the escape codes listed in Chapter 4.

If the enhanced video firmware is active, Right Arrow (Control-U) reads a character from the screen as if it were typed from the keyboard. This is a function of the Monitor's built-in editing capability described in Chapter 4. The enhanced video firmware is discussed in Chapter 6.

Speaker output

The Apple IIc and Apple IIc Plus have a small speaker mounted near the front of the bottom plate of the case. The speaker is connected to a soft switch that toggles; that is, the switch has two states, off and on, and it changes from one to the other each time it is accessed. Table 5-3 describes the speaker output characteristics. The speaker circuit is described in the section "The Speaker," in Chapter 11.

■ **Table 5-3** Speaker output characteristics

Port number	None
Commands	Some programs sound the speaker in response to Control-G
Initial characteristics	Reset routine sounds the speaker
Hardware location	
$C030	Toggle speaker (read only)

Monitor firmware routines

Location	Name	Description
$FBDD	Bell1	Sends a beep to the speaker
$FF3A	Bell	Sends Control-G to the current output

Using the speaker

If you switch the speaker once, by reading $C030, it emits a click; to make longer sounds, access the speaker repeatedly. The switch for the speaker uses memory location $C030. You can make various tones and buzzes with the speaker by using combinations of timing loops in your program.

△ Important You should always use a read operation to toggle the speaker. If you write to this soft switch, it switches twice in rapid succession. △

Monitor firmware support for speaker output

The Monitor supports the speaker with one simple routine, Bell1. A related routine, Bell, supports the current output device—the one that CSW points to.

Bell1

Bell1 (at address $FBDD) makes a beep through the speaker by generating a 1 kHz tone in the speaker for 0.1 second. This routine scrambles the A and X registers.

Bell

The Monitor routine Bell (at location $FF3A) writes a bell control character (ASCII Control-G) to the current output device. This routine leaves the A register holding $87.

Chapter 6 **Video Output**

The primary output device of the Apple IIc is its video display. This chapter describes the way the firmware and memory of the Apple IIc-family computers are used to control the video display. ■

You can use any ordinary color or monochrome video monitor with the Apple IIc or Apple IIc Plus. An ordinary monitor is one that accepts NTSC-compatible composite video. If you use Apple IIc color graphics with a black-and-white monitor, the display appears as black, white, and two shades of gray.

◆ *Note:* **NTSC** stands for *National Television Standards Committee*, a group that formulates TV transmission and reception standards in the USA and several other countries.

If you are using only graphics modes and 40-column text, you can use a television set for your video display. If the TV set has an input connector for composite video, you can connect it directly to your Apple IIc or Apple IIc Plus; otherwise, you must attach an RF video modulator between the computer and the television set.

◆ *Note:* The DB-15 video expansion connector on the back of Apple IIc–family computers can be used for connecting RF modulators, LCD displays, and other devices designed for use with the Apple IIc family. Notice, however, that the video expansion port does not directly support RGB displays, and so cannot be used with the the Apple Color RGB monitor used with the Apple IIGS® computer. The video expansion connector is described in the section "Video Expansion Output" in Chapter 11.

△ Important The Apple IIc family can produce an 80-column text display. However, if you use an ordinary color or black-and-white television set, 80-column text will be too blurry to read. For a clear 80-column display, you must use a high-resolution video monitor with a bandwidth of 14 MHz or greater. △

Table 6-1 lists characteristics of the video output port and points to figures and tables where you can find information about these characteristics.

■ **Table 6-1** Video output port (port 3) characteristics

Characteristic	Where to find more information
Commands	Figure 6-1 "Text Modes and How to Switch Between Them"
Initial characteristics*	Figure 6-1 "Text Modes and How to Switch Between Them"
Video display page locations	Table 6-8 "Video Display Page Locations"
Monitor firmware routines	Table 6-12 "Monitor Firmware Routines"
I/O firmware entry points	Table 6-13 "Port 3 Firmware Protocol Table"

◆ *Note:* On the Apple IIc computers, your program should check the position of the 80/40 switch at $C060 (bit 7 = 1 indicates 40 columns) and set the number of columns accordingly. See Table 6-9. The Apple IIc Plus does not have an 80/40 switch.

Video display specifications

Table 6-2 summarizes the video display's specifications and provides a further guide to other information in this chapter.

■ **Table 6-2** Video display specifications

Display modes	40-column text (map: Figure 6-5) 80-column text (map: Figure 6-6) Lo-Res color graphics (map: Figure 6-7) Hi-Res color graphics (map: Figure 6-8) Double Hi-Res color graphics (map: Figure 6-9)
Text capacity	24 lines by 80 columns (character positions)
Character set	96 ASCII characters (uppercase and lowercase)
Display formats	Normal, inverse, flashing, MouseText (Table 6-4)

■ **Table 6-2** Video display specifications (continued)

Lo-Res graphics	16 colors (Table 6-5): 40 horizontal by 48 vertical (map: Figure 6-7)
Hi-Res graphics	6 colors (Table 6-6): 140 horizontal by 192 vertical (restricted)
	Black and white: 280 horizontal by 192 vertical (map: Figure 6-8)
Double Hi-Res graphics	16 colors (Table 6-7): 140 horizontal by 192 vertical (no restrictions)
	Black and white: 560 horizontal by 192 vertical (map: Figure 6-9)

The video signal produced by the Apple IIc and Apple IIc Plus is NTSC-compatible composite color video available at two places on the back panel of the computer: the RCA-type phono jack and the 15-pin D-type connector. Use the RCA-type phono jack to connect a video monitor, and the DB-15 connector for an external video modulator or other video expansion hardware. The video circuit and connectors are described in the section "The Video Display," in Chapter 11.

Enhanced video firmware

The Apple IIc has two distinct sets of routines in ROM for displaying text on the screen. The standard video firmware supports all the display output available in the Apple II and Apple II Plus computers. The enhanced video firmware supports the additional features provided by an Apple IIe computer with an 80-column card installed. Table 6-3 compares the features of the standard video firmware with those of the enhanced video firmware.

■ **Table 6-3** Comparison of standard and enhanced video firmware

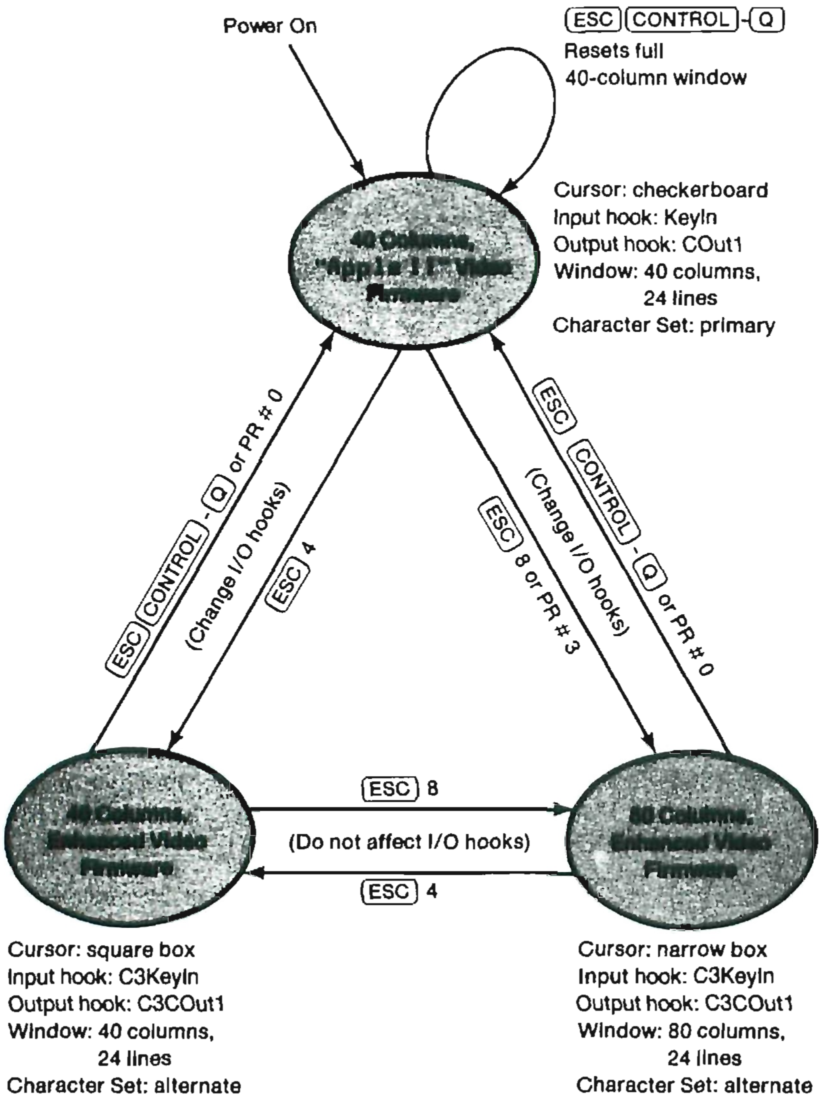
Feature	Standard video firmware	Enhanced video firmware
Character set	Primary	Alternate
Columns	40	40 or 80
Standard input routine	KeyIn	C3KeyIn
Standard output routine	COut1	C3COut1

The primary and alternate character sets are described in the following section, "Text Modes." The standard input and output routines are described in Chapter 4.

Your program can activate the enhanced video firmware by transferring control to address $C300. To deactivate the enhanced video firmware from a program, write a Control-U character using the output subroutine COut.

Figure 6-1 shows the characteristics of the text display modes and how to switch between them from the keyboard.

■ **Figure 6-1** Text modes and how to switch between them



Text modes

In all three text modes shown in Figure 6-1, the Apple IIc–family computers can display all 96 ASCII characters: uppercase and lowercase letters, the ten digits, punctuation marks, and special characters. Each character is displayed in an area of the screen that is seven dots wide by eight dots high. The characters are formed by a dot matrix five dots wide (with a few exceptions, such as underscore), leaving two blank columns of dots between characters in a row. Except for lowercase letters with descenders, the characters are only seven dots high, leaving one blank line of dots between rows of characters.

The normal display has white (or other monochrome color used by your monitor) dots on a dark background. Characters can also be displayed as black dots on a white background; this is called *inverse video.*

Text character sets

The Apple IIc family can display either of two text character sets: the *primary set* and an *alternate set* (Table 6-4). The forms of the characters in the two sets are actually the same, but the available display formats are different. The display formats are

- normal, with white dots on a black screen
- inverse, with black dots on a white screen
- flashing, alternating between normal and inverse

The Apple IIc family can display uppercase characters in all three formats—normal, inverse, and flashing—with the primary character set. Lowercase letters can only be displayed in normal format with the primary character set. This feature makes the primary character set compatible with most software written for the Apple II and Apple II Plus, which can display text in flashing format but don't have lowercase characters.

The alternate character set trades the flashing format for a complete set of inverse characters. With the alternate character set, the Apple IIc family can display uppercase letters, lowercase letters, numbers, and special characters in either normal format or inverse format. It can also display MouseText.

You can select between character sets with the alternate-text soft switch (AltChar), described later in this chapter. Table 6-4 shows the character codes in decimal and hexadecimal for the Apple IIc–family primary and alternate character sets in normal, inverse, and flashing formats.

■ **Table 6-4** Display character sets

	Primary character set		Alternate character set	
Hex values	**Character type**	**Format**	**Character type**	**Format**
\$00–\$1F	Uppercase letters	Inverse	Uppercase letters	Inverse
\$20–\$3F	Special characters	Inverse	Special characters	Inverse
\$40–\$5F	Uppercase letters	Flashing	MouseText	
\$60–\$7F	Special characters	Flashing	Lowercase letters	Inverse
\$80–\$9F	Uppercase letters	Normal	Uppercase letters	Normal
\$A0–\$BF	Special characters	Normal	Special characters	Normal
\$C0–\$DF	Uppercase letters	Normal	Uppercase letters	Normal
\$E0–\$FF	Lowercase letters	Normal	Lowercase letters	Normal

Each character on the screen is stored as one byte of display data. The low-order six bits make up the ASCII code of the character being displayed. The remaining two (high-order) bits select format and the group within ASCII. See Table 5-2 for the ASCII value of each character.

MouseText

The alternate character set contains 32 graphics characters called *MouseText* in place of the primary set's inverse uppercase characters from \$40 through \$5F. These graphics are especially convenient to use with a mouse because they can be generated by character codes instead of groups of Hi-Res byte values, and they can be moved around quickly. To use MouseText characters, do the following:

1. Turn on the enhanced video firmware with PR#3 or 3 Control-P.
2. Set inverse mode: use the System Monitor Inverse command or put \$3F in location \$32, or print Control-O using C3COut1.
3. Turn on MouseText with PRINT CHR\$(27); or pass \$1B to COut in the A register.

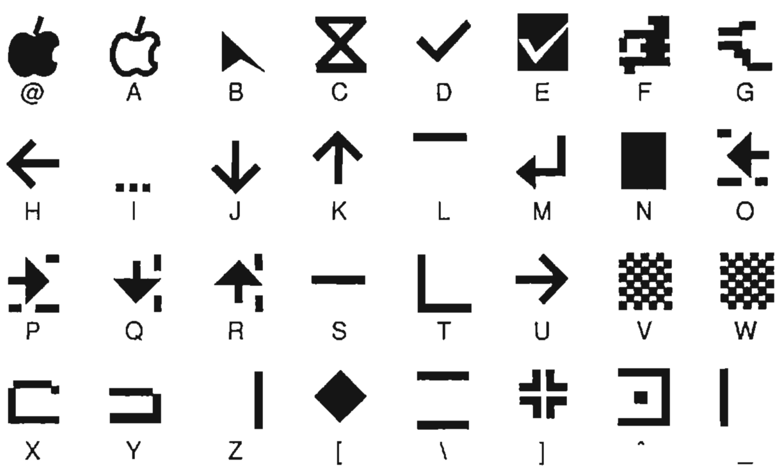
4. Print the uppercase letter (or other ASCII character in the range $40 through $5F: @ [\ ] ^ or _) that corresponds to the MouseText character you want.
5. Turn off MouseText with PRINT CHR$(24); or pass $18 to COut in the accumulator.
6. Set normal mode: use the Normal command or put $FF in location $32, or print a Control-N.

Here is a sample Applesoft program that prints all the MouseText characters:

```
10 D$=CHR$(4)
20 PRINT PRINT D$;"PR#3"
30 INVERSE
40 PRINT CHR$(27);"   ABCDEFGHIJKLMNOPQRSTUVWXYZ[]^_";
50 PRINT CHR$(24);
60 NORMAL
```

MouseText characters and their corresponding ASCII characters are shown in Figure 6-2.

■ **Figure 6-2** MouseText characters



40-column versus 80-column text

The Apple IIc family has two text display modes: 40-column and 80-column. The number of dots in each character does not change, but the characters in 80-column mode are only half as wide as the characters in 40-column mode. Compare the two displays in Figure 6-3. On an ordinary color or black-and-white television set, the narrow characters in the 80-column display blur together; you must use the 40-column mode to display text on a television set.

△ Apple IIc Plus — There is no 80/40 switch on the Apple IIc Plus computer; 80/40-column switching is strictly software controlled. △

■ **Figure 6-3** 40-column and 80-column text with alternate character set

```
]LIST 0,100

10  REM  APPLESOFT CHARACTER DEMO
20  TEXT : HOME
30  PRINT : PRINT "Applesoft Char
     acter Demo"
40  PRINT : PRINT "Which characte
     r set—"
50  PRINT : INPUT 'Primary (P) or
     Alternate (A) ?"A$
60  IF LEN (A$) < 1 THEN 50
65  LET A$ = LEFT$ (A$,1)
70  IF A$ = "P" THEN  POKE 49166,
     0
80  IF A$ = "A" THEN  POKE 49167,
     0
90  PRINT : PRINT "...printing th
     e same line, first"
100  PRINT ' in NORMAL, then INVE
     RSE, then FLASH:": PRINT

]
```

```
]LIST 0,100

10  REM  APPLESOFT CHARACTER DEMO
20  TEXT : HOME
30  PRINT ; PRINT "Applesoft Character Demo"
40  PRINT : PRINT "Which character set—"
50  PRINT : INPUT "Primary (P) or Alternate (A) ?";A$
60  IF LEN (A$) < 1 THEN 50
65  LET A$ = LEFT$ (A$,1)
70  IF A$ = "P" THEN  POKE 49166,0
80  IF A$ = "A" THEN  POKE 49167,0
90  PRINT : PRINT "...printing the same line, first"
100  PRINT " in NORMAL, then INVERSE, then FLASH:": PRINT

]
```

Graphics modes

The Apple IIc family can produce color video graphics in any of three different modes:

- Lo-Res graphics, 48 rows by 40 columns
- Hi-Res graphics, 192 rows by 280 columns
- Double Hi-Res graphics, 192 rows by 560 columns

Each graphics mode treats the screen as a rectangular array of spots. Normally, your programs use the features of some high-level language to draw graphics dots, lines, and shapes on the screen; this section describes the way the resulting graphics data are stored in memory.

Lo-Res graphics

The Apple IIc family displays an array of 48 rows by 40 columns of colored blocks in the Lo-Res graphics mode. Each block can be any one of sixteen colors, including black and white. On a black-and-white monitor or television set, these colors appear as black, white, and two shades of gray. There are no blank dots between blocks; adjacent blocks of the same color merge to make a larger shape.

The Lo-Res graphics display data are stored in the same part of memory as the data for the 40-column text display. Each byte contains data for two Lo-Res graphics blocks. The two blocks are displayed one atop the other in a display space the same size as a 40-column text character, seven dots wide by eight dots high.

Half a byte—four bits, or one nibble—is assigned to each graphics block. Each nibble can have a value from 0 to 15, and this value determines which one of sixteen colors appears on the screen. The colors and their corresponding nibble values are shown in Table 6-5. In each byte, the low-order nibble sets the color for the top block of the pair, and the high-order nibble sets the color for the bottom block. Thus, a byte containing the hexadecimal value $D8 produces a brown block atop a yellow block on the screen.

■ **Table 6-5** Lo-Res graphics colors

Nibble value		
Dec	**Hex**	**Color**
0	$00	Black
1	$01	Magenta
2	$02	Dark blue
3	$03	Purple
4	$04	Dark green
5	$05	Gray 1
6	$06	Medium blue
7	$07	Light blue
8	$08	Brown
9	$09	Orange
10	$0A	Gray 2
11	$0B	Pink
12	$0C	Light green
13	$0D	Yellow
14	$0E	Aquamarine
15	$0F	White

Note: Colors may vary, depending on the adjustment of the monitor or television set.

As explained earlier in this chapter, the text display and the Lo-Res graphics display use the same area in memory. Your programs should usually clear this part of memory when they change display modes, but you can store data as text and display them as graphics, or vice versa. All you have to do is change the mode switch, without changing the display data. Mode switches are shown in Table 6-9. This usually produces meaningless jumbles on the display, but some programs have used this technique to good advantage for producing complex Lo-Res graphics displays quickly.

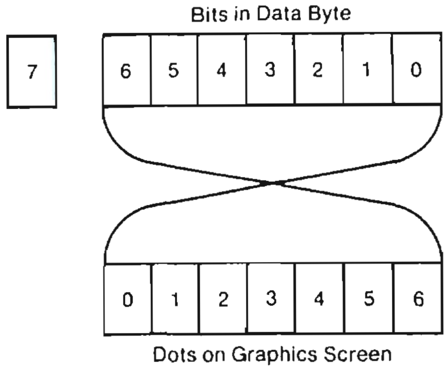
Hi-Res graphics

In the Hi-Res graphics mode, the Apple IIc family displays an array of colored dots in 192 rows and 280 columns. The colors available are black, white, purple, green, orange, and blue, although the colors of the individual dots are limited, as described below, by the color of adjacent dots. Adjacent dots of the same color merge to form a continuous colored area.

Hi-Res graphics display data are stored in either of two 8192-byte areas in memory. These areas are called *Hi-Res Page 1* and *Page 2;* think of them as display data buffers.

The Apple IIc–family Hi-Res graphics display is bit-mapped: each dot on the screen corresponds to a bit in the computer's memory. The 7 low-order bits of each display byte control a row of 7 adjacent dots on the screen, and 40 adjacent bytes in memory control a row of 280 (7 times 40) dots. The eighth bit (the most significant) of each byte is not displayed; it selects one of two color sets, as described below. The least significant bit of each byte is displayed as the leftmost dot in a row of 7, followed by the next-least significant bit, and so on, as shown in Figure 6-4.

■ **Figure 6-4** Hi-Res display bits



There is a simple correspondence between bits in memory and dots on the screen on a black-and-white monitor. A dot is white if the bit controlling it is on (1), and the dot is black if the bit is off (0). On a black-and-white television set, pairs of dots merge together; alternating black and white dots merge to a continuous gray.

A dot whose controlling bit is off (0) is black on an NTSC color monitor or a color television set. If the bit is on, the dot is white or a color, depending on its position, the dots on either side, and the setting of the high-order bit of the byte. Call the leftmost column of dots column 0, and assume (for the moment) that the high-order bits of all the data bytes are off (0). If the bits that control them are on, dots in even-numbered columns, 0, 2, 4, and so forth, are purple, and dots in odd-numbered columns are green—but only if the dots on either side are black. If two adjacent dots are both on, they are both white.

You select the other two colors, blue and orange, by turning the high-order bit (bit 7) of a data byte on (1). The colored dots controlled by a byte with the high-order bit on are either blue or orange: the dots in even-numbered columns are blue, and the dots in odd-numbered columns are orange (again, only if the dots on either side are black). Within each horizontal line of seven dots controlled by a single byte, you can have black, white, and one pair of colors. To change the color of any dot to one of the other pair of colors, you must change the high-order bit of its byte, which affects the colors of all seven dots controlled by the byte.

In brief, Hi-Res graphics displayed on a color monitor or television set are made up of colored dots, according to the following rules:

- Dots in even-numbered columns can be black, purple, or blue.
- Dots in odd-numbered columns can be black, green, or orange.
- If adjacent dots in a row are both on, they are both white.
- The colors in each row of seven dots controlled by a single byte are either purple and green, or blue and orange, depending on whether the high-order bit is off (0) or on (1).

These rules are summarized in Table 6-6. The blacks and whites are numbered to remind you that the high-order bit is different.

■ **Table 6-6** Hi-Res graphics colors

Bits 0–6	**Bit 7 off**	**Bit 7 on**
Adjacent columns off	Black 1	Black 2
Even columns on	Purple	Blue
Odd columns on	Green	Orange
Adjacent columns on	White 1	White 2

Note: Colors may vary, depending on the adjustment of the monitor or television set.

The peculiar behavior of the Hi-Res colors reflects in part the way NTSC color television works. The dots that make up the Apple IIc–family video signal are spaced to coincide with the frequency of the color subcarrier used in the NTSC system. Alternating on and off dots at this spacing cause a color monitor or TV set to produce color, but two or more on dots together do not. For more information on how the video circuits work, see the section "The Video Display," in Chapter 11.

Double Hi-Res graphics

The horizontal resolution of Double Hi-Res graphics is 560 dots per line, with 192 lines. Double Hi-Res graphics maps the low-order 7 bits of the bytes in the two Double Hi-Res graphics pages. A Double Hi-Res page is made up of a 8192-byte page in main memory and an equivalent page having the same address in auxiliary memory. In most cases, only the first Double Hi-Res graphics page is used.

The bytes in the main-memory and auxiliary-memory pages are displayed in exactly the same manner as the characters in 80-column text: of each pair of identical addresses, the auxiliary-memory byte is displayed first, and the main-memory byte is displayed second. A dot whose controlling bit is off (0) is black when displayed.

Unlike Hi-Res color, Double Hi-Res color has no restrictions on which colors can be adjacent. Color is determined by any four adjacent dots along a line. Think of a four-dot-wide window moving across the screen: at any given time, the color displayed corresponds to the 4-bit value from Table 6-7 that corresponds to the window's position (Figure 6-9). Effective horizontal resolution with color is 140 (560 divided by 4).

Table 6-7 describes the data values used to produce colors in Double Hi-Res graphics. To use the table, divide the column number by four and use the remainder to find the correct column: *ab0* is a byte residing in auxiliary memory corresponding to a remainder of 0 (byte 0, 4, 8, and so on), *mb1* is a byte residing in main memory corresponding to a remainder of 1 (byte 1, 2, 9 and so on), and similarly for *ab2* and *mb3*.

■ **Table 6-7** Double Hi-Res graphics colors

Color	ab0	mb1	ab2	mb3	Repeated bit pattern
Black	$00	$00	$00	$00	0000
Magenta	$08	$11	$22	$44	0001
Brown	$44	$08	$11	$22	0010
Orange	$4C	$19	$33	$66	0011
Dark green	$22	$44	$08	$11	0100
Gray 1	$2A	$55	$2A	$55	0101
Green	$66	$4C	$19	$33	0110
Yellow	$6E	$5D	$3B	$77	0111
Dark blue	$11	$22	$44	$08	1000
Purple	$19	$33	$66	$4C	1001
Gray 2	$55	$2A	$55	$2A	1010
Pink	$5D	$3B	$77	$6E	1011
Medium blue	$33	$66	$4C	$19	1100
Light blue	$3B	$77	$6E	$5D	1101
Aqua	$77	$6E	$5D	$3B	1110
White	$7F	$7F	$7F	$7F	1111

Note: Colors may vary, depending on the adjustment of the monitor or television set.

You might want to display 40-column text with a program that uses Double Hi-Res graphics so that the text can be read on a television set. Because the 80COL soft switch must be switched on in order to display Double Hi-Res graphics, there is no way to used mixed mode to display 40-column text with Double Hi-Res graphics. If you do not want to define your own Double Hi-Res character set, you can send text to the Apple IIc text screen locations in memory and then toggle the text screen on to display the text.

To set up the text screen in memory, use a PR#3 command (or its equivalent) to initialize the enhanced video firmware. Because the soft switches are still set to display Double Hi-Res graphics, the image on the screen does not change; however, text sent to COUT is stored in the correct locations in memory for display on the text screen. Be careful not to exceed 40 characters per line. When you have finished sending the text to COUT, send a Control-Q command to COUT. This command switches off the 80COL soft switch, so that Double Hi-Res graphics is automatically switched off and the 40-column text screen is switched on, displaying the text you sent to memory through COUT.

When you are ready to return to Double Hi-Res graphics, send a Control-R command to COUT. This command switches the 80COL soft switch back on, and the screen returns to the Double Hi-Res graphics display, which is still preserved in the appropriate locations in memory.

When you switch from graphics to text modes, the screen momentarily goes to Hi-Res mode before going to 40-column text mode. If you switch first to text mode, the screen goes to 80-column text before you can switch to 40-column text. If the brief appearance of the wrong display causes a problem for your program, you can synchronize the change with the vertical blanking interval of the display.

Mixed-mode displays

Any of the graphics displays can have four lines of text, either 40-column or 80-column, at the bottom of the screen. Graphics displays with text at the bottom are called *mixed-mode displays*. To use them, the TEXT switch must be off (read $C050) and the MIXED switch on (read $C053).

△ Important — You cannot use a mixed-mode display to display 40-column text with Double Hi-Res graphics. △

To determine what appears where in mixed-mode displays, refer to Figures 6-5 through 6-9 later in this chapter. See the bottom sixth of the appropriate text display (Figure 6-5 or 6-6) and the upper five-sixths (down to the heavy horizontal line) in the appropriate graphics display (Figures 6-7 to 6-9).

Display pages

The Apple IIc–family computers use data stored in specific areas in memory to generate video displays. These areas, called *display pages*, serve as buffers where your programs can put data to be displayed. Each byte in a display buffer controls an object—a character, a colored block, or a group of adjacent dots—at a certain location on the display, depending on the current display mode.

The 40-column-text and Lo-Res graphics modes use two display pages of 1024 bytes each. These are called *text Page 1* and *text Page 2*, and they are located at $0400 through $07FF and $0800 through $0BFF in main memory. Normally, only Page 1 is used, but you can put text or graphics data into Page 2 and switch between displays. Either page can be displayed as 40-column text, Lo-Res graphics, or mixed-mode (four lines of text at the bottom of a graphics display).

The 80-column text mode displays twice as much data as the 40-column mode—1920 bytes—but it cannot switch pages when the enhanced video firmware is active. The 80-column text display uses a combination page made up of text Page 1 in main memory plus another page in auxiliary memory. This additional memory is *not* the same as text Page 2—in fact, it is text Page 1X, and it occupies the same address space as text Page 1 (see Figure 2-9). The built-in firmware I/O routines described in Chapter 4 take care of this extra addressing automatically; that is one reason to use these routines for all normal text output.

△ Important — The built-in video firmware always displays text Page 1. You cannot write text to text Page 2 with the built-in firmware. △

The Hi-Res graphics mode also has two display pages, but each page is 8192 bytes long. In the 40-column text and Lo-Res graphics modes each byte controls a display area seven dots wide by eight dots high. In Hi-Res graphics mode each byte controls an area seven dots wide by one dot high. Thus, a Hi-Res display requires eight times as much data storage as a Lo-Res display, as shown in Table 6-8.

The Double Hi-Res graphics mode interleaves the two Hi-Res pages (Pages 1 and 1X) in exactly the same way as 80-column text mode interleaves the text pages: column 0 and all subsequent even-numbered columns come from the auxiliary page; column 1 and all subsequent odd-numbered columns come from the main page.

■ **Table 6-8** Video display page locations

Display mode	Display page	Lowest address Hex	Dec	Highest address Dex	Dec
40-column text, Lo-Res graphics	1	$0400	1024	$07FF	2047
	2*	$0800	2048	$0BFF	3071
80-column text	1	$0400	1024	$07FF	2047
	2*	$0800	2048	$0bFF	3071
Hi-Res graphics	1	$2000	8192	$3FFF	16383
	2	$4000	16384	$5FFF	24575
Double Hi-Res graphics	1†	$2000	8192	$3FFF	16383
	2†	$4000	6384	$5FFF	24575

* This is not supported by firmware; for instructions on how to switch pages, refer to the next section "Display Mode Switching."

† See "Double Hi-Res Graphics," earlier in this chapter.

Display mode switching

Table 6-9 shows the reserved locations for the soft switches that control the different display modes. The column of the table labeled *Action* indicates what to do to activate or read a switch setting: *R* means read the location, *W* means write anything to the location, *R/W* means read or write, and *R7* means read the location and then check bit 7.

Table 6-10 lists the display modes that the firmware can set up automatically. In the 40-column modes, the contents of the standard I/O links KSW and CSW (described in Chapter 4) determine whether the enhanced video firmware features are available or not. The firmware also takes care of setting or clearing AltChar.

Table 6-11 lists other display modes available but not supported by firmware. For modes that display Page 2 with the 80Col switch on, your program may have to turn 80Store off after the firmware has turned it on.

Double Lo-Res shows on the display screen when HiRes is off and both 80Col and DHiRes are on. It is the low-resolution graphics equivalent of 80-column text, and it uses the same map (Figure 6-6), giving you 48 rows of 80 blocks.

The IOUDis ($C07E) switch must be on to allow you to use locations $C05E and $C05F to change DHiRes. The firmware in fact leaves it on—and your program should, too—unless it wants to use locations $C05E and $C05F to change mouse values (explained in Chapter 9).

Table 6-9 Display soft switches

Name	Action	Hex	Function
AltChar	W	$C00E	Off: Display text using primary character set
AltChar	W	$C00F	On: Display text using alternate character set
RdAltChar	R7	$C01E	Read AltChar switch (1 = on)
80Col	W	$C00C	Off: Display 40 columns
80Col	W	$C00D	On: Display 80 columns
Rd80Col	R7	$C01F	Read 80Col switch (1 = on)
80Store	W	$C000	Off: Cause Page2 on to select auxiliary RAM
80Store	W	$C001	On: Allow Page2 to switch main RAM areas
Rd80Store	R7	$C018	Read 80Store switch (1 = on)
Page2	R/W	$C054	Off: Select Page 1
Page2	R/W	$C055	On: Select Page 1X (80Store on) or 2
RdPage2	R7	$C01C	Read Page2 switch (1 = on)
Text	R/W	$C050	Off: Display graphics or (if Mixed on) mixed
Text	R/W	$C051	On: Display text
RdText	R7	$C01A	Read Text switch (1 = on)
Mixed	R/W	$C053	Off: Display only text or only graphics
Mixed	R/W	$C054	On: (If Text off) display text and graphics
RdMixed	R7	$C01B	Read Mixed switch (1 = on)
HiRes	R/W	$C057	Off: (If Text off) display Lo-Res graphics
HiRes	R/W	$C058	On: (If Text off) display Hi-Res or (if DHiRes on) Double Hi-Res graphics
RdHiRes	R7	$C01D	Read HiRes switch (1 = on)
IOUDis	W	$C07E	On: Disable IOU access for addresses $C058 to $C05F; enable access to DHiRes switch†

Table 6-9 Display soft switches (continued)

Name	Action	Hex	Function
IOUDis	W	$C07F	Off: Enable IOU access for addresses $C058 to $C05F; disable access to DHiRes switch*†
RdIOUDis	R7	$C07E	Read IOUDis switch (1 = off)†
DHiRes	R/W	$C05E	On: (If IOUDis on) turn on Double Hi-Res
DHiRes	R/W	$C05F	Off: (If IOUDis on) turn off Double Hi-Res
RdDHiRes	R7	$C07F	Read DHiRes switch (1 = off)†

* The firmware normally leaves IOUDis on. See also the following footnote.

† Reading or writing any address in the range $C070–$C07F also triggers the paddle timer and resets the VBL interrupt flag (see Chapter 9).

■ **Table 6-10** Display modes supported by firmware, including Applesoft

				Switches					
Display col/res	**Type**	**Page**	**80Col**	**80Store**	**Page2**	**Text**	**Mixed**	**HiRes**	**DHiRes**
40-column	Text	1	Off		Off	On	Off	Off	Off
80-column	Text	1	On	*		On			
Lo-Res	Graphics	1	Off		Off	Off	Off	Off	Off
40/Lo-Res	Mixed	1	Off		Off	Off	On	Off	
80/Lo-Res	Mixed	1	On	*	Off	Off	On	Off	Off
Hi-Res	Graphics	1	Off		Off	Off	Off	On	
Hi-Res	Graphics	2	Off		On	Off	Off	On	
40/Hi-Res	Mixed	1	Off		Off	Off	On	On	
80/Hi-Res	Mixed	1	On	*	Off	Off	On	On	Off

* 80Store is set by the firmware when 80Col is turned on.

■ **Table 6-11** Other display modes

Display col/res	Type	Page	80Col	80Store	Page2	Text	Mixed	HiRes	DHiRes
			Switches						
40-column	Text	2	Off		On	On			
80-column		2	On	Off	On	On			
Low-res	Graphics	2	Off		On	Off	Off	Off	
40/Lo-Res	Mixed	2	Off		On	Off	On	Off	
80/Lo-Res	Mixed	2	On	Off	On	Off	On	Off	Off
Dbl Lo-Res	Graphics	1	On	•	Off	Off	Off	Off	On
Dbl Lo-Res	Graphics	2	On	Off	On	Off	Off	Off	On
80/Dbl Lo-Res	Mixed	1	On	•	Off	Off	On	Off	On
80/Dbl Lo-Res	Mixed	2	On	Off	On	Off	On	Off	On
40/Hi-Res	Mixed	2	Off		On	Off	On	On	
80/Hi-Res	Mixed	2	On	Off	On	Off	On	On	Off
Dbl Hi-Res	Graphics	1	On	•	Off	Off	Off	On	On
Dbl Hi-Res	Graphics	2	On	Off	On	Off	Off	On	On
80/Dbl Hi-Res	Mixed	1	On	•	Off	Off	On	On	On
80/Dbl Hi-Res	Mixed	2	On	Off	On	Off	On	On	On

• 80Store is set by the firmware when 80Col is turned on and must be turned off to use the second 80-column or Double Hi-Res page. This means that you cannot use firmware routines such as COut when displaying Page 2 modes not supported by firmware.

For example, to switch to mixed 80-column and Double Hi-Res display Page 1, you can use these instructions in your program:

`STA`	`$C00D`	Turns on 80Col; firmware then turns on 80Store.
`LDA`	`$C054`	Turns off Page2; you could also have done a STA.
`STA`	`$C050`	Turns off Text; that is, turns on graphics mode.
`STA`	`$C053`	Turns on Mixed; it works now that Text is off.
`STA`	`$C057`	Turns on HiRes; it works now that Text is off.
`STA`	`$C07E`	Makes sure IOUDis is on so you can access DHiRes.
`LDA`	`$C05E`	Turns on DHiRes; it works now that IOUDis is on.

Display page maps

You should never have to store directly into display memory. Most high-level languages let you write statements that control the text and graphics displays. Similarly, if you are programming in assembly language, you should use the display features of the built-in I/O firmware.

▲ **Warning** Never call any firmware with 80Col on or with 80Store and Page2 both on. If you do, the firmware will not function properly. As a general rule, always leave Page2 off. ▲

All the different display modes use the same basic addressing scheme: characters or graphics bytes are stored as rows of 40 contiguous bytes, but the rows themselves are not stored at locations corresponding to their locations on the display. Instead, the display address is transformed so that three rows that are eight rows apart on the display are grouped together and stored in the first 120 locations of each block of 128 bytes ($80 hex). For example, the first 128-byte block contains the data for rows 0, 8, and 16. The next 128-byte block contains data for rows 1, 9, and 17, and so on.

The display memory maps are shown in Figures 6-5 through 6-9. For a full description of the way the Apple IIc–family hardware handles display memory, see Chapter 11.

Hi-Res graphics data are stored in much the same way as text, but there are eight times as many bytes to store, because eight rows of dots occupy the same space on the display as one row of characters.

The first 1024 bytes of the Hi-Res display page contain the first row of dots from *each* of the 24 groups of eight rows of dots. The second 1024 bytes of the Hi-Res display page contain the second row of dots from *each* group of eight rows of dots, and so on for all eight rows of all the groups. This fills up the 8192 bytes of the Hi-Res display page.

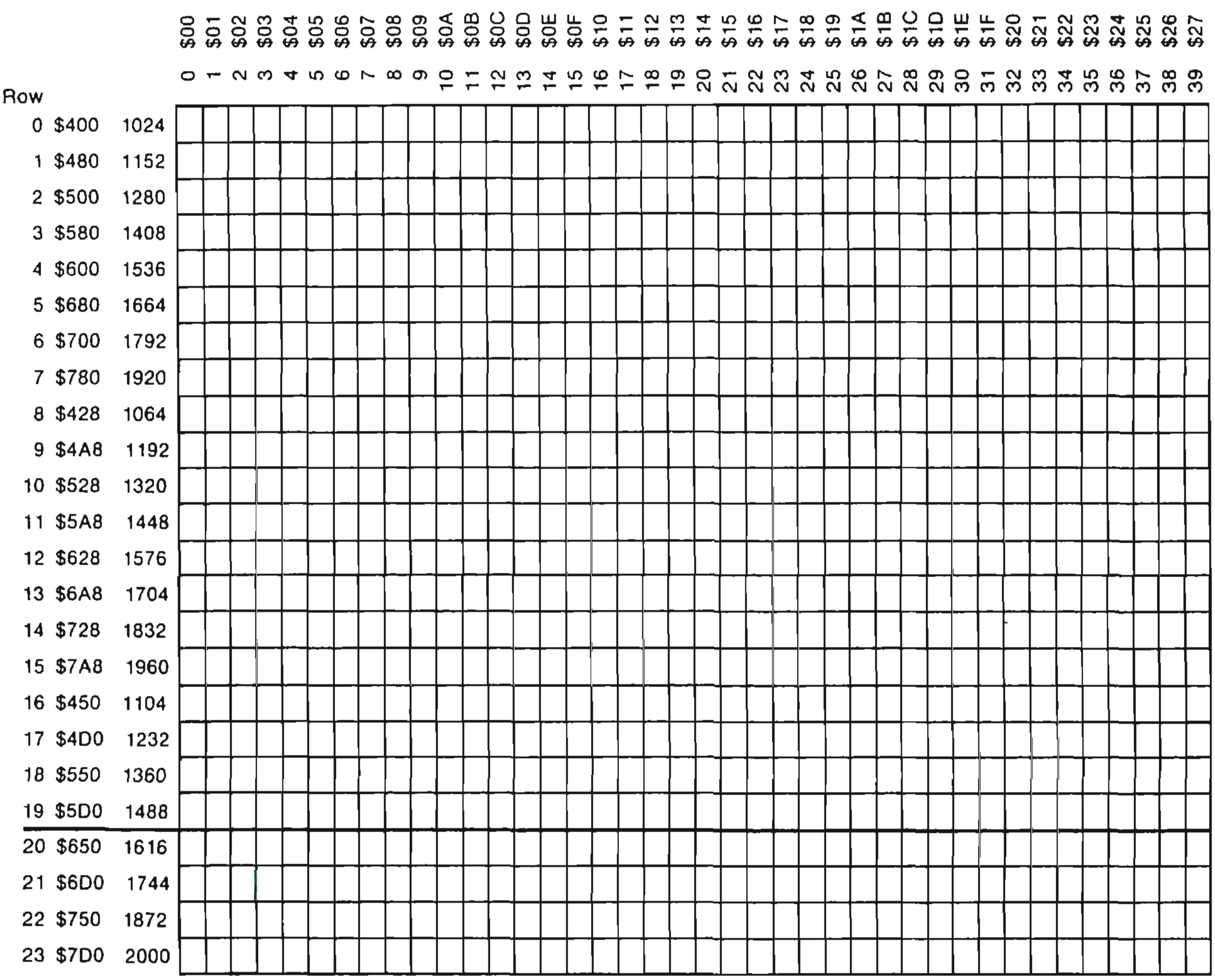
The display maps show addresses only for each Page 1. To obtain addresses for text or Lo-Res graphics Page 2, add 1024 ($0400); to obtain addresses for Hi-Res Page 2, add 8192 ($2000).

The 80-column display works a little differently. Half of the data are stored in the normal text Page 1 memory, and the other half are stored in the *auxiliary* memory text Page 1. The display circuitry fetches bytes from the same address in both memory areas simultaneously and displays them sequentially: first the byte from the auxiliary memory, then the byte from the main memory. The characters in the even-numbered columns of the display are stored (starting with column 0) in main memory, and the characters in the odd-numbered columns of the display are stored (starting with column 1) in main memory.

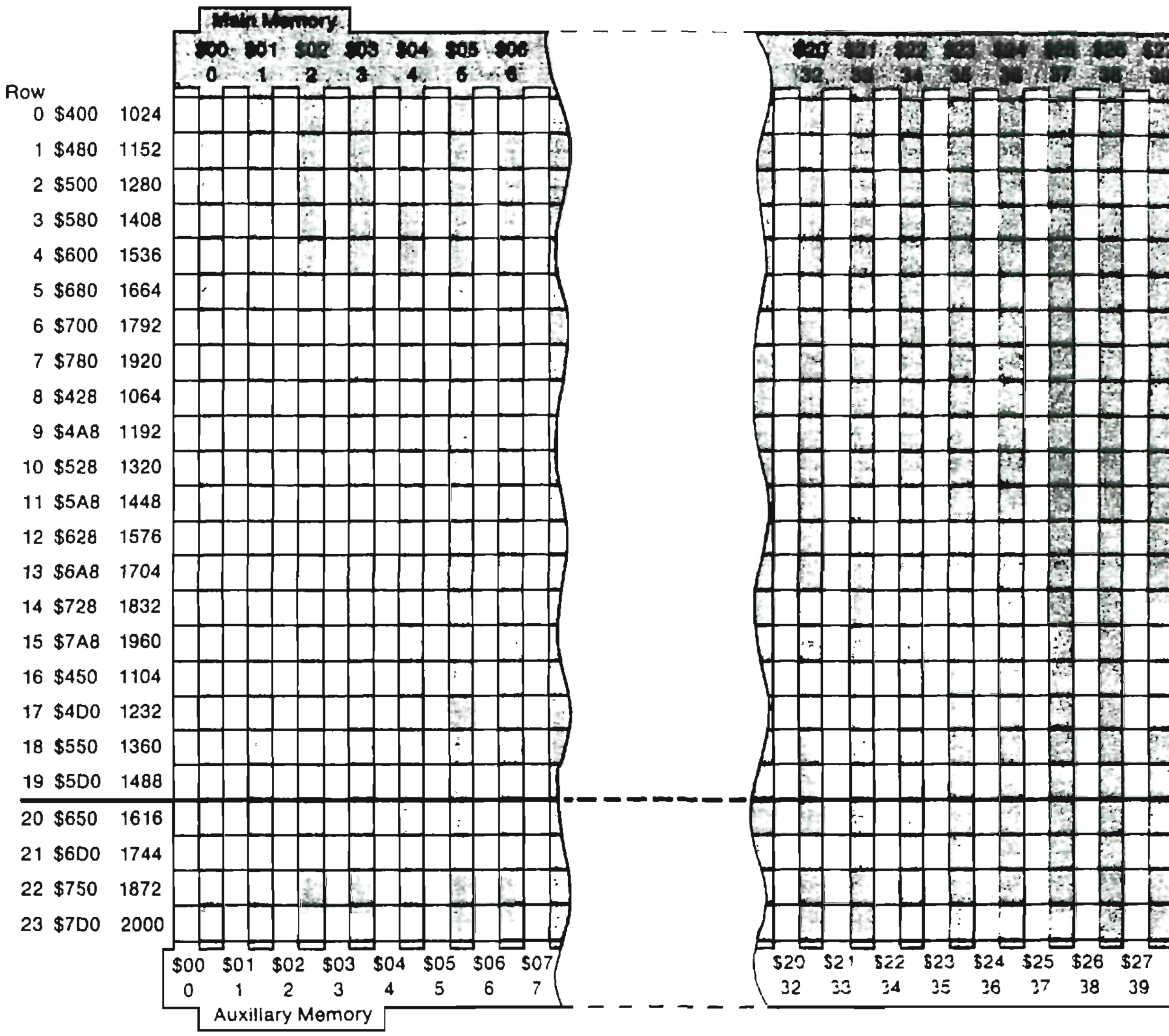
To store display data in auxiliary memory, first turn on the 80Store soft switch by writing to location $C001. With 80Store on, the page-select switch Page2 selects between the portion of the 80-column display stored in Page 1 of main memory and the portion stored in the auxiliary memory. To select auxiliary memory, turn the Page2 soft switch on by reading or writing at location $C055.

The Double Hi-Res graphics display stores information in the same way as Hi-Res graphics, except there is an auxiliary memory location as well as a main memory location corresponding to each address. The two sets of display information are interleaved in a manner similar to the interleaving of two 40-column displays to create an 80-column text display (see Figure 6-9).

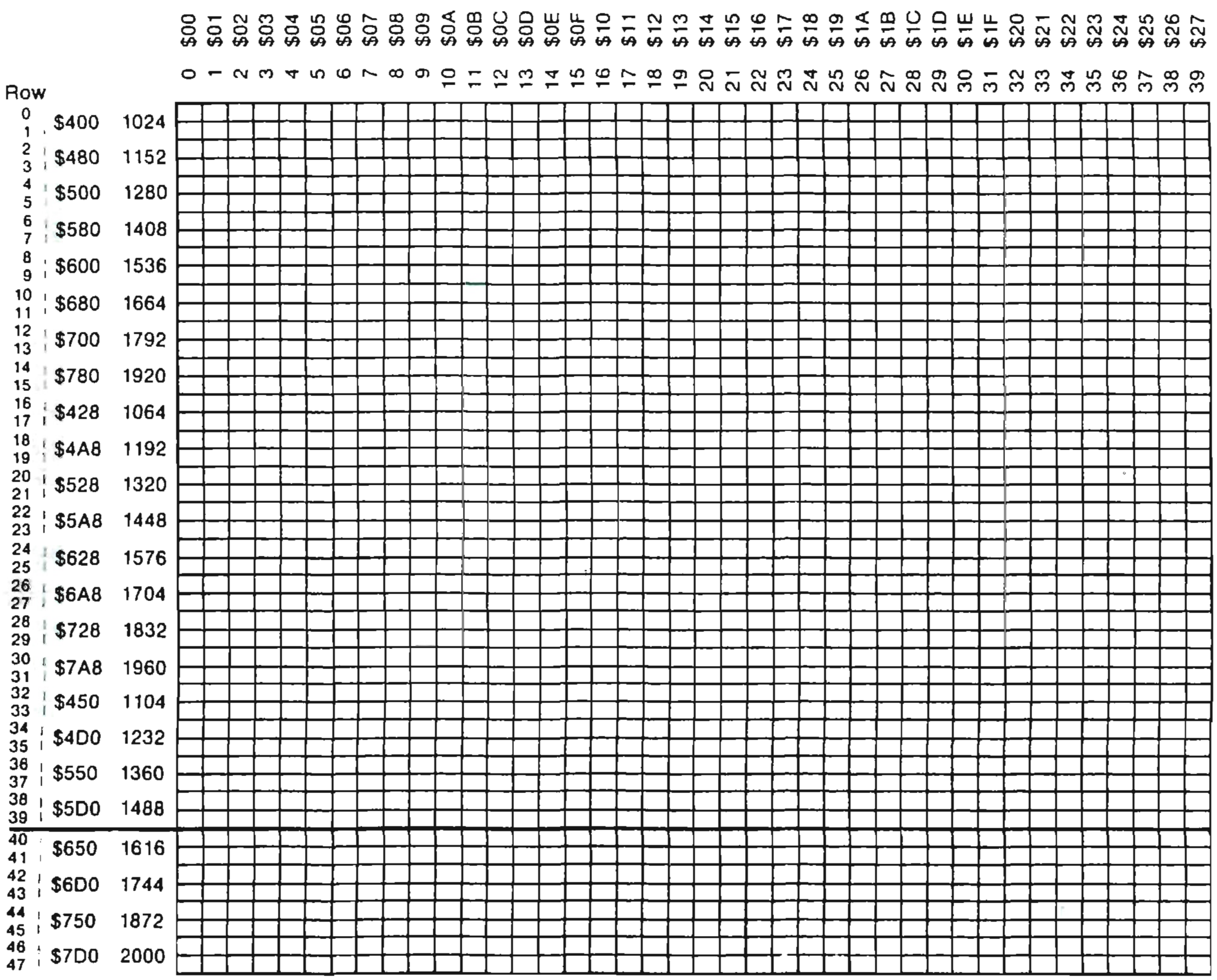
■ **Figure 6-5** Map of 40-column text display



■ **Figure 6-6** Map of 80-column text display



■ **Figure 6-7** Map of Lo-Res graphics display



■ **Figure 6-8** Map of Hi-Res graphics display

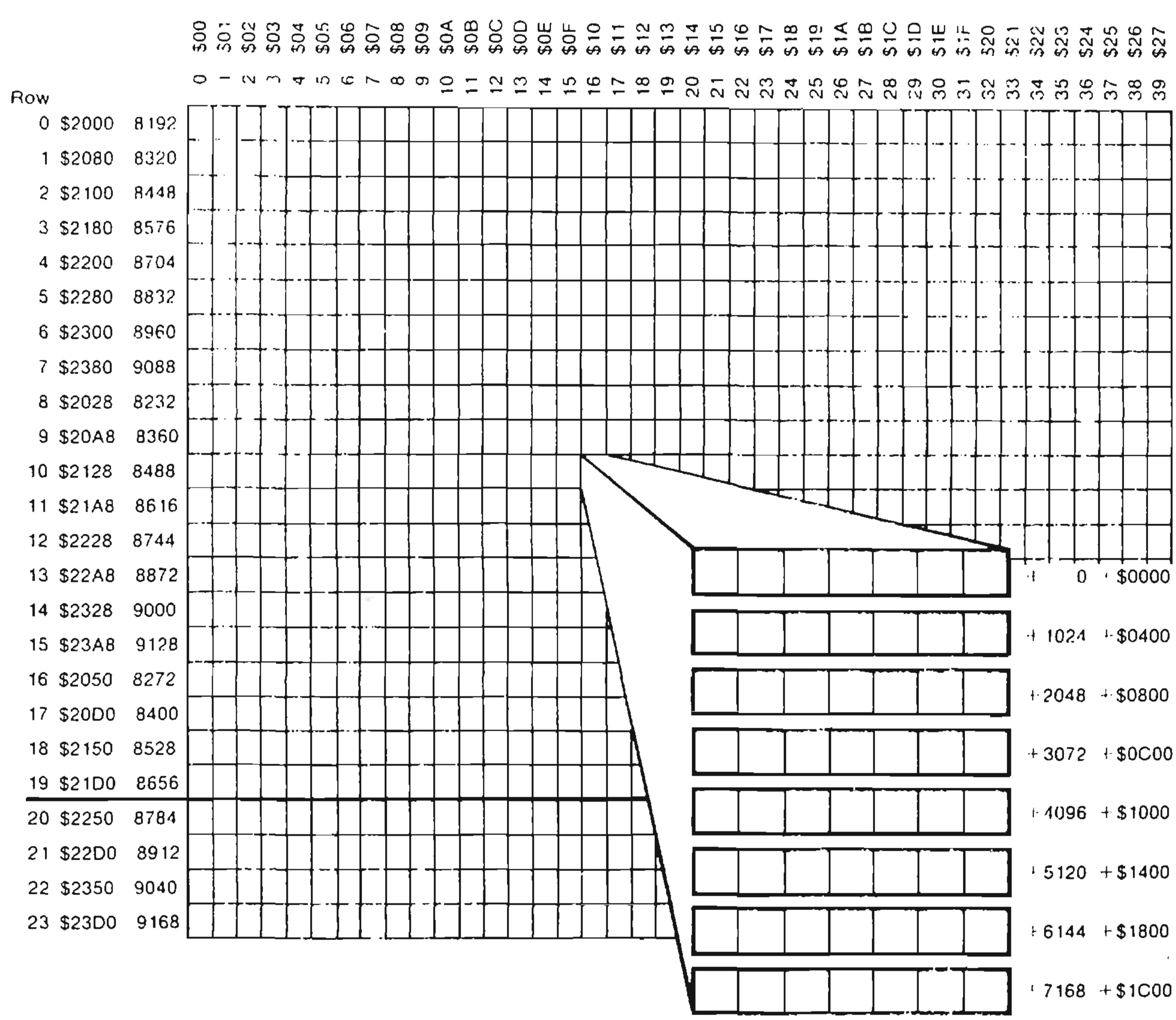
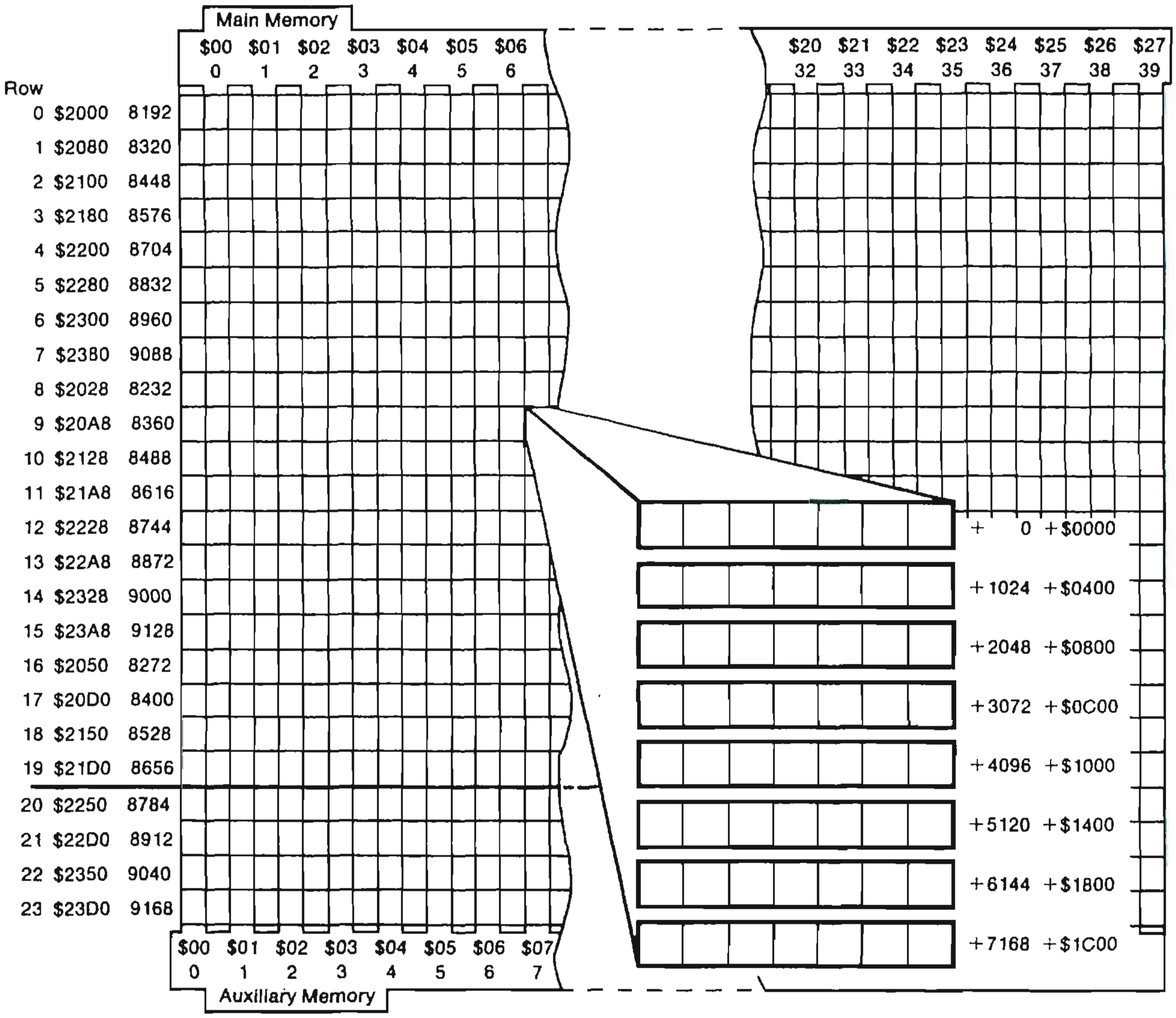


Figure 6-9 Map of Double Hi-Res graphics display



Monitor support for video display output

Table 6-12 summarizes the addresses and functions of the video display support routines the Monitor provides. These routines are described in Appendix F. COut and Cout1 are described in more detail in Chapter 4.

■ **Table 6-12** Monitor firmware routines

Name	Location	Description
ClrEOL	$FC9C	Clears to end of line from current cursor position
ClEOLZ	$FC9E	Clears to end of line using contents of Y register as cursor position
ClrEOP	$FC42	Clears to bottom of window from current cursor position
ClrScr	$F832	Clears the Lo-Res screen
ClrTop	$F836	Clears top 40 lines of Lo-Res screen
COut	$FDED	Calls output routine whose address is stored in CSW (normally COut1, Chapter 4); displays character stored in A register
COut1	$FDF0	Displays a character on the screen (Chapter 4)
CROut	$FD8E	Generates a carriage return character
CROut1	$FD8B	Clears to end of line, then generates a carriage return character
HLine	$F819	Draws a horizontal line of blocks on Lo-Res resolution screen
Home	$FC58	Clears the window and puts cursor in upper-left corner of window
Plot	$F800	Plots a single Lo-Res block on the screen
PrBl2	$F94A	Sends 1 to 256 blank spaces to the output device whose address is in CSW
PrByte	$FDDA	Prints a hexadecimal byte
PrErr	$FF2D	Sends ERR and Control-G to the output device whose output routine address is inCSW
PrHex	$FDE3	Prints 4 bits as a hexadecimal number
PrntAX	$F941	Prints contents of A and X registers in hexadecimal
SCRN	$F871	Reads color value of a Lo-Res block on the screen
SetCol	$F864	Sets the color for plotting in Lo-Res mode
VTabZ	$FC24	Sets cursor vertical position (setting CV at location $25 does not change vertical position until a carriage return)
VLine	$F828	Draws a vertical line of Lo-Res blocks

I/O firmware support for video output

Apple IIc–family video firmware conforms to the I/O firmware protocol described in Chapter 4. However, it does not support windows other than the full 80-by-24 window in 80-column mode, and the full 40-by-24 window in 40-column mode.

The video (port 3) protocol table is shown in Table 6-13.

■ **Table 6-13** Port 3 firmware protocol table

Address	Value	Description
$C30B	$01	Generic signature byte of firmware cards
$C30C	$88	80-column card device signature
$C30D	$ii	$C3ii is entry point of initialization routine (PInit)
$C30E	$rr	$C3rr is entry point of read routine (PRead)
$C30F	$ww	$C3ww is entry point of write routine (PWrite)
$C310	$ss	$C3ss is entry point of the status routine (PStatus).

PInit

The PInit routine initializes the enhanced video firmware. PInit does the following:

- sets a full 80-column window
- sets 80Store ($C001)
- sets 80Col ($C00D)
- switches on AltChar ($C00F)
- clears the screen; places cursor in upper-left corner
- displays the cursor

PRead

The PRead routine reads a character from the keyboard and places it in the A register with the high bit cleared. It also puts a 0 in the X register to indicate no error before returning to the calling program.

PWrite

The PWrite routine displays a character on the screen or executes a control-character command, as shown in Table 6-14. Your program should place a character in the A register with its high bit cleared before calling PWrite. PWrite does the following:

- turns the cursor display off
- if the character in the A register is a control character, carries out control functions as shown in Table 6-14
- if the character in the A register is not a control character, checks the inverse flag (location $32) and turns the high bit on for normal display or off for inverse display
- if the character is not a control character, displays the character at the current cursor position and advances the cursor
- if the cursor is advanced past the end of a line, does carriage return but not line feed

When PWrite has completed this, it

- turns the cursor display back on (if it was not intentionally turned off by a Control-F command)
- puts a 0 in the X register to indicate no error and returns to the calling program

■ **Table 6-14** Video control functions

Control-	Hex	Function
E or e	$05	Turns cursor on (enables cursor display)
F or f	$06	Turns cursor off (disables cursor display)
G or g	$07	Sounds bell (beeps)
H or h	$08	Moves cursor left one column; if cursor was at beginning of line, moves it to end of previous line
J or j	$0A	Moves cursor down one row; scrolls if needed
K or k	$0B	Clears to end of screen
L or l	$0C	Clears screen; moves cursor to upper-left position on screen
M or m	$0D	Moves cursor to column 0
N or n	$0E	Displays subsequent characters in normal video; characters already on display are unaffected
O or o	$0F	Displays subsequent characters in inverse video; characters already on display are unaffected
V or v	$16	Scrolls screen up one line; clears bottom line
W or w	$17	Scrolls screen down one line; clears top line
Y or y	$19	Moves cursor to upper-left (home) position on screen
Z or z	$1A	Clears entire line that cursor is on
| or \	$1C	Moves cursor right one column; if at end of line, does Control-M
} or]	$1D	Clears to end of the line the cursor is on, including current cursor position; does not move cursor
^ or 6	$1E	GOTOxy: Initiates a GOTOxy sequence; interprets the next two characters as x+32 and y+32, respectively
_	$1F	If not at top of screen, moves cursor up one line

PStatus

The PStatus routine returns the I/O status of the enhanced video firmware. A program that calls PStatus must first put a request code in the A register: either a 0 (meaning "is the enhanced video firmware ready for output?") or a 1 (meaning "is there any input—that is, has a character been typed on the keyboard?"). PStatus returns with the reply in the c flag (the carry flag): 0 (no) or 1 (yes). If the request was not 0 or 1, PStatus returns with a 3 in the X register, indicating an illegal operation. If the request was a 0 or 1, PStatus returns with a 0 in the X register indicating no error.

Chapter 7 **Serial Port I/O**

The Apple IIc–family computers have two serial I/O ports. Although these ports are electrically identical, they are set up differently when they are initialized by the reset routine. Port 1 is initialized as an output port for RS-232 type devices, such as printers and plotters. Port 2 is set up to operate as a serial communications port for modems and similar devices.

Either port can be reconfigured by software after initialization. Guidelines for making changes for each port are given in this chapter.

The serial port back-panel connectors and interface circuitry are described in Chapter 11. ■

△ Important — Although the Apple IIc–family serial ports are similar to the Apple IIe Super Serial Card, there are important differences. Refer to Appendix D for a summary of these differences. △

△ Apple IIc Plus — Although the Apple IIc Plus serial port connectors are identical to AppleTalk connectors, the Apple IIc Plus does not contain firmware support for AppleTalk. △

Sending commands to the serial ports

The two following sections describe how to send commands to serial ports 1 and 2. Table 7-1 describes all the commands used by the Apple IIc–family serial ports. Both ports use the same set of commands.

Using serial port 1

You can access the firmware from BASIC by issuing Control-D (if DOS or ProDOS is in RAM) and PR#1. Subsequent output is directed to the printer (or other device) connected to serial port 1.

To direct Pascal output to the printer, you can use either #6: or PRINTER:.

Your non-Pascal programs can also access the port by changing the value of CSW (see Chapter 4).

Table 7-1 lists the commands you can use with serial port 1, either from a program or from the keyboard, after you issue PR#1. Each command must be preceded by Control-I (the command character). As soon as you issue the command character, the serial port firmware displays a flashing question mark cursor to indicate it is awaiting a command. You do not have to press Return after commands that you have entered from the keyboard, or send the return character from your program if it is sending commands to the port. You can type more than one command on a line, but each must be preceded by the command character.

The serial port 1 command character is set as Control-I when the Apple IIc or Apple IIc Plus is turned on. You can change it to a different control character by sending the current control character followed immediately by the new control character you want. This is useful if you want to be able to send Control-I to the printer without firmware intervention. For example, to change the command character from Control-I to Control-V, send Control-I Control-V either from the keyboard or from a program. (Control-V and Control-W are the recommended substitute control characters.) To change the command character back again, send Control-V Control-I. Don't slip any spaces between the control characters that you send.

▲ **Warning** Do not use Control-A, -B, -C, -H, -J, -L, -M, or -Y: Apple IIc–family firmware may intercept these control characters, causing unpredictable results. ▲

The following are examples of valid commands and command sequences. These examples all show commands being entered from the keyboard, but your programs can send the characters just as well. Remember to issue a PR#1 before starting to send commands to serial port 1.

◆ *Note:* The spaces shown in the following commands are for clarity only; don't include spaces in the actual command string.

To echo output to the display screen:

Control-I I

To set line width 72, disable line feed, and echo:

Control-I K Control-I 7 2 N

To change control character to Control-V:

Control-I Control-V Return

To set up the serial port to allow sending Control-I as part of a character stream:

Control-V {command} Return

Using serial port 2

You can access the firmware from BASIC in the usual way—that is, by issuing Control-D (if DOS or ProDOS is in RAM) followed by IN#2 or PR#2. Subsequent input and output are routed through the modem (or other device) connected to serial port 2.

△ Important In terminal mode, the modem port commands listed in Table 7-2 must follow Control-D and IN#2 (not PR#2) and the command character (which is usually Control-A). △

To transfer files to the modem under Pascal, specify REMOUT: or #8:. To transfer files from the modem under Pascal, specify REMIN: or #7:.

Table 7-1 lists the commands you can use with serial port 2, either from a program or from the keyboard, after you issue IN#2. Each command must be preceded by Control-A (the command character). As soon as you issue the command character, the serial port firmware displays a flashing question mark cursor to indicate it is awaiting a command. If you press Return, you get the current video cursor again. You do not have to press Return (or send a return character) after commands. You can type more than one command on a line, but each must be preceded by the command character.

When the Apple IIc or Apple IIc Plus is turned on, the serial port 2 command character is defined as a Control-A. You can change it to a different control character by typing the current control character followed immediately by the new control character you want. This is useful if you want to be able to send Control-A to the output device without firmware intervention.

For example, to change the command character from Control-A to Control-V, send Control-A Control-V either from the keyboard or from a program. (Control-V and Control-W are the recommended substitute control characters.) To change the command character back again, send Control-V Control-A.

▲ **Warning** Do not use Control-B, -C, -H, -I, -J, -L, -M, or -Y: Apple IIc–family firmware may intercept these control characters, causing unpredictable results. ▲

The following are examples of valid commands and command sequences. These examples show commands being entered from the keyboard, but your programs can send the characters just as well.

To enable echo to the screen:

Control-A I

To send a break character to a remote device:

Control-A B

To change the control character to Control-V (for example, so you can send Control-A as part of a character stream):

Control-A Control-V Control-V(command)

Serial port command set

△ Original IIc — The following commands (from Table 7-1) are not available for the original Apple IIc: C F M X. △

■ **Table 7-1** Serial port commands

<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td>nnn</td><td>Sets new line width of nnn (from 1 through 255). This command must be followed by N (see nnnN) or by a carriage return.</td></tr>
<tr><td>nnB</td><td>Sets transmission rate to value corresponding to nn:
<table>
<tr><th>nn</th><th>Baud</th><th>nn</th><th>Baud</th><th>nn</th><th>Baud</th></tr>
<tr><td>1</td><td>50</td><td>6</td><td>300</td><td>11</td><td>3600</td></tr>
<tr><td>2</td><td>75</td><td>7</td><td>600</td><td>12</td><td>4800</td></tr>
<tr><td>3</td><td>110 (109.92)</td><td>8</td><td>1200</td><td>13</td><td>7200</td></tr>
<tr><td>4</td><td>135 (134.58)</td><td>9</td><td>1800</td><td>14</td><td>9600</td></tr>
<tr><td>5</td><td>150</td><td>10</td><td>2400</td><td>15</td><td>1920</td></tr>
</table></td></tr>
<tr><td>C*</td><td>When enabled, this command causes a carriage return character to be sent automatically whenever the column count exceeds the printer line width. The command is normally enabled.</td></tr>
<tr><td>nD</td><td>Sets data format to values corresponding to n:
<table>
<tr><th>n</th><th>Data bits</th><th>Stop bits</th><th>n</th><th>Data bits</th><th>Stop bits</th></tr>
<tr><td>0</td><td>8</td><td>1</td><td>4</td><td>8</td><td>2</td></tr>
<tr><td>1</td><td>7</td><td>1</td><td>5</td><td>7</td><td>2</td></tr>
<tr><td>2</td><td>6</td><td>1</td><td>6</td><td>6</td><td>2</td></tr>
<tr><td>3</td><td>5</td><td>1</td><td>7</td><td>5</td><td>2</td></tr>
</table></td></tr>
<tr><td>F*</td><td>When this command is enabled, your Apple IIc–family computer accepts data from the keyboard as well as from the serial port. You can use this to disable the keyboard before receiving or sending data to prevent accidental keystrokes from disrupting the data flow. Be sure that your program reenables the keyboard when the data transfer is complete. This command is available only from BASIC and is normally enabled.</td></tr>
</table>

■ **Table 7-1** Serial port commands (continued)

Command	Description
I	Echoes printer output on the screen.
K	Disables automatic line feed after carriage return.
L†	Generates line feed after carriage return. Normally, this command is enabled. Disabling it has the same effect as the K command.
M*	When this command is enabled, all incoming line feed characters that are not immediately preceded by carriage return characters are masked (removed from the data stream). Normally this command is enabled.
*nnn*N	Changes line width to *nnn* (from 1 through 255; *nnn* is optional); does not echo printer output on the screen. *Note:* 0N does not disable automatic generation of carriage return; to do so, use Z command, put 0 directly in location $0579, or use the system utilities disk.
*n*P	Sets parity corresponding to *n*:
Q	Quit terminal mode.
R	Resets port and exits from serial port firmware.
S	Sends a 233-millisecond BREAK character (used with some printers to synchronize with serial ports).
T	Starts terminal mode. Use after IN#2 only. If followed by PR#2, the computer echoes input to output.

Parity values for the *n*P command:

n	Parity	*n*	Parity
0	None	4	None
1	Odd	5	MARK (1)
2	None	6	None
3	Even	7	SPACE (0)

■ **Table 7-1** Serial port commands (continued)

Command	Description
X*	When enabled, this command turns on the XON/XOFF protocol: the computer looks for the XOFF ($13) character and responds by halting transmission until an XON ($11) is received. Normally this command is disabled.
Z	Zaps (ignores) further command characters until Control-Reset or PR#1. Does not format output or insert carriage returns into output stream.
Control-T	When issued by a remote device after the IN#1 or IN#2 command is already in effect, this command puts the Apple IIc or Apple IIc Plus in terminal mode. This command cannot be issued locally; it has the same effect as Control-A T typed locally.
Control-Q	When issued by a remote device after the IN#1 or IN#2 command is already in effect, this command causes the Apple IIc or Apple IIc Plus to quit terminal mode. This command cannot be issued locally; it has the same effect as Control-A Q typed locally.

* Not available on the original Apple IIc. This command can be toggled: If you follow the command with E (with no intervening space), the command is enabled. If you follow the command with D (with no intervening space), the command is disabled.

† The L command is available for all versions of the Apple IIc, but cannot be toggled on the original Apple IIc.

◆ *Note:* The commands are letter-type commands, *not* control characters.

Port 1

This section describes the characteristics and use of port 1 when configured as a printer port. Table 7-2 summarizes the characteristics of port 1. If you change port 1 to a communication port (like port 2), refer to the section "Port 2," later in this chapter, for a description of its use.

■ **Table 7-2** Serial port 1 characteristics

Port number	Serial port 1.
Commands	Keyboard command: PR#1. BASIC command: PR#1. Monitor command: 1 Control-P (does not work if there is an operating system in RAM). All other commands: See Table 7-1.
Initial characteristics	See "Characteristics of Port 1 at Startup."
Hardware page locations	See Table 7-3.
Monitor firmware routines	None.
I/O firmware entry points	See Table 7-4.
Use of screen holes	See Table 7-5.
Use of other pages	None.

Characteristics of port 1 at startup

After power-up, the printer firmware sets the following configuration:

- 9600 baud
- eight data bits, no parity bits, two stop bits
- 80-column line width; no echo to display screen
- firmware supplies line feed after carriage return
- command character is set to Control-I

These values are stored in the auxiliary memory screen holes (Table 7-5). You can change some of these settings from the keyboard by typing `PR#1`, the command character, and one of the commands listed in Table 7-1. How port characteristics change as a result of various activities is described under "Changing Port 1 Characteristics" later in this chapter.

Hardware page locations for port 1

Table 7-3 lists for serial port 1 the addresses and bit assignments of its hardware registers on page $C0. The registers are internal to a 6551 ACIA; their bit assignments are described in Chapter 11.

▲ **Warning** This table is for your information only. To avoid having problems with the system, you should *never* try to directly access the hardware. Instead, use the firmware routines from the Apple IIc–family ROM described in Appendix F. ▲

■ **Table 7-3** Port 1 hardware page locations

Location	Description
$C090–$C097	Reserved
$C098	ACIA transmit/receive data register
$C099	ACIA status register
$C09A	ACIA command register
$C09B	ACIA control register
$C09C–$C09F	Reserved

I/O firmware support for port 1

Table 7-4 lists the locations and values of the I/O firmware protocol table. This standardized protocol is available for use by any application program. Chapter 4 describes how to use this protocol.

■ **Table 7-4** Port 1 I/O firmware protocol

Address	Value	Description
$C105	$38	Pascal ID byte.
$C107	$18	Pascal ID byte.
$C10B	$01	Generic signature byte of firmware cards.
$C10C	$31	Same ID as for Super Serial Card.
$C10D	$ii	$C1ii is entry point of initialization routine (PInit).
$C10E	$rr	$C1rr is entry point of read routine (PRead).
$C10F	$ww	$C1ww is entry point of write routine (PWrite).
$C110	$ss	$C1ss is entry point of the status routine (PStatus).
$C111	non-zero	No optional routines.

Screen hole locations for port 1

Table 7-5 lists the screen hole locations that serial port 1 uses. The auxiliary memory locations are reserved for *startup* value settings which are listed and interpreted in the table.

■ **Table 7-5** Port 1 screen hole locations

Location	Description
Auxiliary memory screen holes (firmware loads values at power up)	
$0478	$9E (ACIA control register: 8 data + 2 stop bits, 9600 baud)
$0479	$0B (ACIA command register: no parity)
$047A	$40 (flags: no echo, auto LF after CR, type of serial port)
	Bit **Interpretation**
	7 Echo output on display (0 = no echo)
	6 Generate LF after CR (0 = no LF)
	5-1 Always = 0 (reserved)
	0 1 = communication port; 0 = serial printer port.
$047B	$50 (printer width: 80 columns)

■ **Table 7-5** Port 1 screen hole locations (continued)

Location	Description
	Bit **Interpretation**
	7–0 Printer width (0 = do not insert CR)
Main memory screen holes	
$0479	Reserved
$04F9	Reserved
$0579	Printer width (1–255; 0 = disable formatting)
$05F9	Temporary storage location
$0679	Bit 7 = 1 while the firmware is parsing a command string
$06F9	Current command character (initially Control-I)
$0779	Bit 7 = 1 if echo to display is on; bit 6 = 1 if firmware is to generate a LF after CR
$07F9	Current printer column

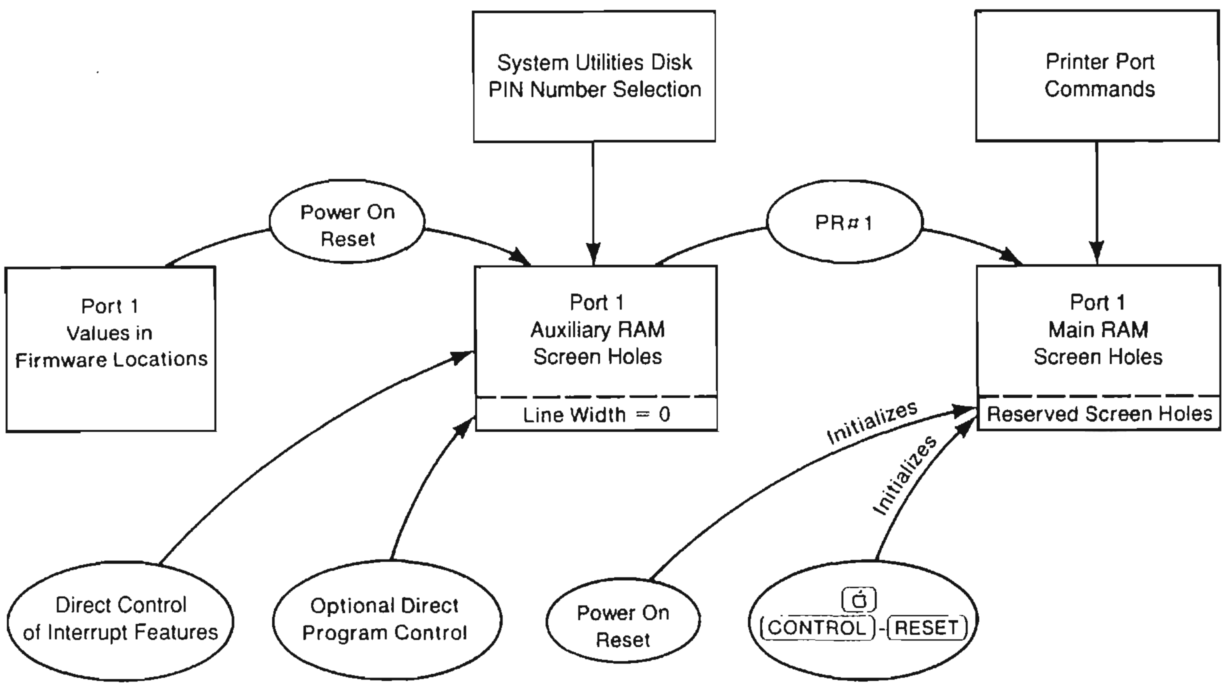
Changing port 1 characteristics

Figure 7-1 is a diagram of where the port characteristics are stored and moved under different circumstances. You can see the following from the figure:

- When the power is first turned on, the reset routine moves the predefined startup characteristics listed earlier in this section from ROM into the auxiliary memory screen holes listed in Table 7-5.
- If you specify new characteristics using the system utilities, the system utilities software changes the values in the auxiliary memory screen holes. Your programs can do the same thing.
- The values stored in the auxiliary memory screen holes are affected by power-on reset, but not by either Command-Control-Reset or a simple Control-Reset. This feature is provided so that a port that has been reconfigured will remain that way while some other program (such as an application program) is started up. (See Figure 7-1.)
- PR#1 causes the firmware to move the characteristics stored in the auxiliary-memory screen holes into the main-memory screen holes.

- A program can change values in the main-memory screen holes directly. However, the only value guaranteed to be in the same place for the entire Apple II series is the line length in main-memory location $0579.
- The firmware uses the port as it is defined in the main-memory screen holes at any given time. You should use the commands listed in Table 7-1 to change them.

■ **Figure 7-1** Diagram of port 1 characteristics storage



Data format and communication rate

Serial data transfer consists of a string of 1's and 0's sent down a wire at a prearranged rate of transmission, measured in baud.

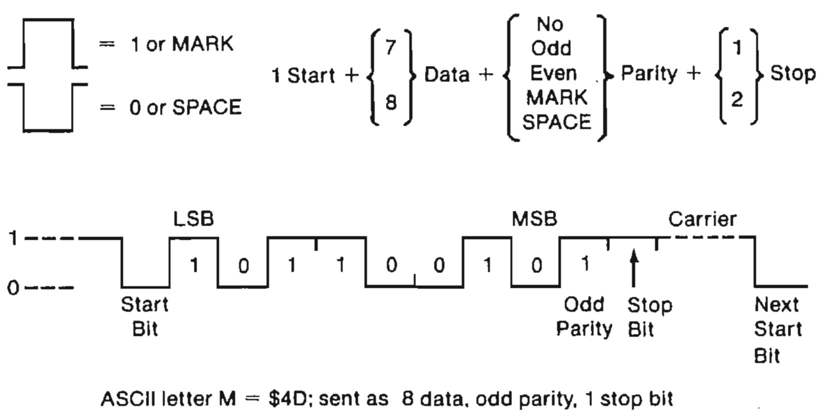
Before transfer begins, both sender and receiver look for a continuous value of 1: this is called the carrier (see Figure 7-2). When the value goes to 0, the receiver presumes it is a start bit—that is, the bit that designates the beginning of a character of data. If it lasts longer than a bit could possibly last, it is considered a BREAK signal, which some printers use for synchronization.

If the first 0 proves to be a bit, it is interpreted as the start bit. Next come the seven or eight data bits (six is seldom used with computers), low-order bit first. If parity is on, it comes next in the message. Finally, one or two stop bits appear. The stop bits have a value of 1, like the carrier. The next start bit begins transfer of the next character of data.

The parity bit provides a simple check of data validity. Odd parity means the sender counts the number of 1's among the data bits, and sends the appropriate parity bit to make the total number of 1's odd. With even parity, the sender adds the appropriate parity bit to make the total number of 1 bits even. MARK parity is always a 1 bit; SPACE parity is always a 0. The receiver can then check that the parity bit is correct.

If the transmission rate is 300 baud and the data format is one start bit plus seven data bits plus one parity bit plus one stop bit (totaling ten bits transmitted for each byte of data sent), then the actual transfer rate is about 30 characters per second.

■ **Figure 7-2** Data format



Carriage return and line feed

If you are using a typewriter and you push the carriage all the way to the right (in other words, position the printing mechanism at the left margin), you have performed a carriage return. On the other hand, turning the platen so the paper moves to the next line (or using the index key on an electric typewriter) is called a line feed. Most typewriters perform a line feed automatically after a carriage return, and so the two seem to be one—but they are not.

Carriage return and line feed are separate ASCII codes. Carriage return is sometimes denoted *CR;* it is ASCII code $0D (13). Line feed, sometimes denoted *LF,* is ASCII code $0A (10). Down Arrow on all Apple IIc–family keyboards generates a LF.

Some printers can supply a line feed automatically after detecting a carriage return; others cannot. If the printer does not supply a line feed after a carriage return and it is not supplied in the data stream, the printer keeps printing over and over on the same line. On the other hand, if both the printer and the computer firmware supply LF after CR, double line-spacing results.

If the print head keeps moving too far to the right across the page and then prints many characters on top of one another on the right, then the firmware should be instructed to furnish CR after a certain line width has been reached. If the printer prints too short a line before moving to the next line, then probably the firmware is using too small a line width.

If the printer misses characters at the beginning of each line but otherwise prints correctly, there is probably not enough time for the print mechanism to return to the left margin in response to CR. You must use a lower transmission rate with such a printer.

Sending special characters

If you want to send special characters (control characters) to the printer without having them intercepted and executed by the computer's firmware, use the Z command (see Table 7-1). If the only special character that causes a problem is the command character (normally Control-I for port 1), you can change just the command character instead of using the zap (Z) command. If you use the zap command, the firmware does no formatting; that is, it does not check line width or insert carriage returns or line feeds. This may be necessary to send graphics to a printer or plotter.

Displaying output on the screen

You can display printer output on the screen, but if the printer line width exceeds the 40 or 80 columns you have selected for display, you should turn off video display.

Port 2

This section describes the characteristics and use of port 2 when configured as a communication port. Table 7-6 summarizes the characteristics of port 2. If you change port 2 to a printer port (like port 1), refer to the section "Port 1," earlier in this chapter, for a description of its use.

■ **Table 7-6** Serial port 2 characteristics

Port number	Serial port 2.
Commands	Keyboard commands: IN#2 before Table 7-1 commands, IN#2 to accept port 2 input, PR#1 to echo input to printer, PR#2 to echo input back to port 2. BASIC commands: same. Monitor command: 2 Control-P (does not work if there is an operating system in RAM). All other commands: see Table 7-1.
Initial characteristics	See "Characteristics of Port 2 at Startup."
Hardware page locations	See Table 7-7.
Monitor firmware routines	None.
I/O firmware entry points	See Table 7-8.
Use of screen holes	See Table 7-9.
Use of other pages	In terminal mode, firmware uses auxiliary memory locations \$0800–\$087F to store keyboard input, and \$0880–\$08FF as a serial input buffer.

Characteristics of port 2 at startup

After power-up, the firmware sets the following configuration:

- 300 baud
- eight data bits, no parity bits, one stop bit
- firmware does not supply line feed after carriage return
- firmware does not insert carriage returns into output stream
- firmware does not echo output to the display screen
- command character is set to Control-A

These values are stored in the auxiliary memory screen holes (see Table 7-9). You can change some of these settings from the keyboard using the command character followed by one of the commands listed in Table 7-1. How port characteristics change as a result of various activities is described later in this chapter.

If you change any of these values using keyboard commands or commands from a program, subsequent accesses to the port firmware (even by another program) use the new settings instead of the power-up values. This allows you to change the settings once at system startup and get the desired configuration for subsequent uses.

Hardware page locations for port 2

Table 7-7 lists for serial port 2 the addresses of its hardware registers on page $C0. The registers are internal to a 6551 ACIA; their bit assignments are described in Chapter 11.

▲ **Warning** This table is for your information only. To avoid having problems with the system, you should never try to directly access the hardware. Instead, use the firmware routines from the Apple IIc–family ROM described in Appendix F. ▲

■ **Table 7-7** Port 2 hardware page locations

Location	Description
$C0A0–$C0A7	Reserved
$C0A8	ACIA transmit/receive data register
$C0A9	ACIA status register
$C0AA	ACIA command register
$C0AB	ACIA control register
$C0AC–$C0AF	Reserved

I/O firmware support for port 2

Table 7-8 lists the values in the I/O firmware protocol table for serial port 2. This standardized protocol is available for use by any application program. Chapter 4 describes how to use this protocol.

■ **Table 7-8** Port 2 I/O firmware protocol

Address	Value	Description
$C205	$38	Pascal ID byte.
$C207	$18	Pascal ID byte.
$C20B	$01	Generic signature byte of firmware cards.
$C20C	$31	Same ID as for Super Serial Card.
$C20D	$ii	$C2ii is entry point of initialization routine (PInit).
$C20E	$rr	$C2rr is entry point of read routine (PRead).
$C20F	$ww	$C2ww is entry point of write routine (PWrite).
$C210	$ss	$C2ss is entry point of the status routine (PStatus).
$C211	nonzero	No optional routines.

Screen hole locations for port 2

Table 7-9 lists the screen hole locations that serial port 2 uses. Note that the auxiliary memory locations are reserved for *startup* value settings.

■ **Table 7-9** Port 2 screen hole locations

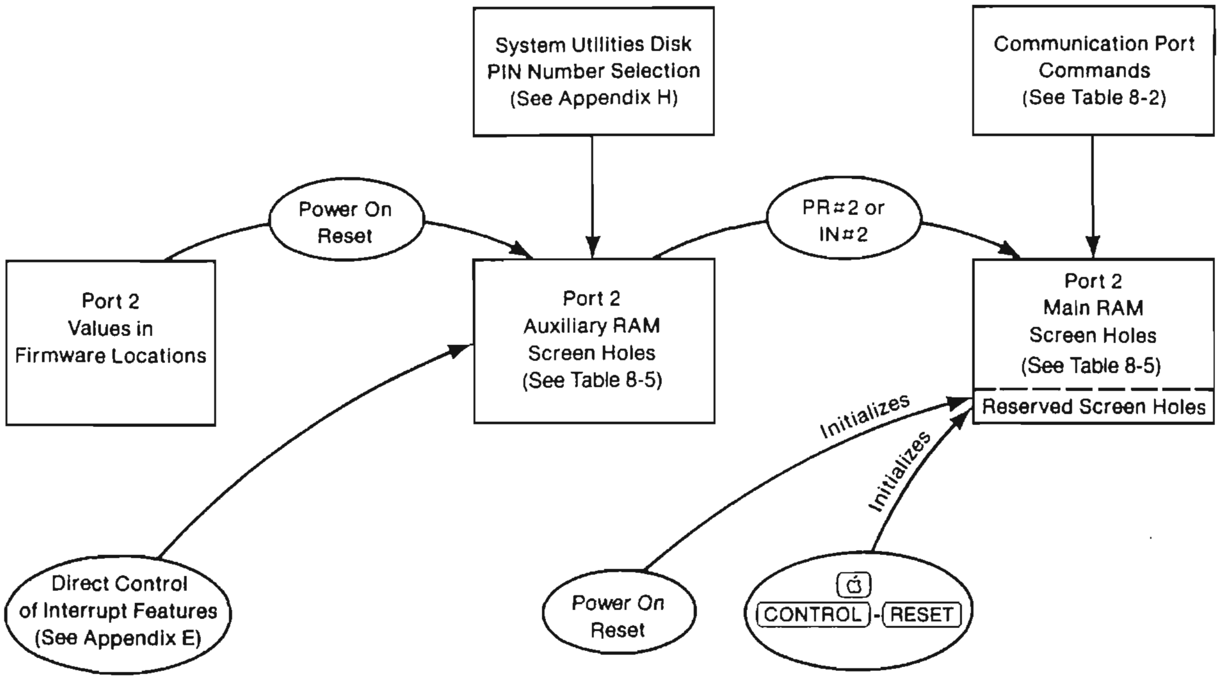
Location	Description
Auxiliary memory screen holes (firmware loads values at power-up)	
$047C	$16 (ACIA control register: 8 data + 1 stop bit, 300 baud)
$047D	$0B (ACIA command register: no parity)
$047E	$01 (flags: no echo, no auto LF after CR, communication port) **Bit Interpretation** 7 Echo output on display (0 = no echo) 6 Generate LF after CR (0 = no LF) 5-1 Always = 0 (reserved) 0 1 = communication port; 0 = serial printer port
$047F	$00 (line length: do not add any CR to output stream) **Bit Interpretation** 7-0 Line length (0 = do not insert CR)
Main memory screen holes	
$047A	Reserved
$04FA	Reserved
$057A	Line length (1–255; 0 = disable formatting)
$05FA	Temporary storage location
$067A	Bit 7 = 1 if and only if the firmware is currently parsing a command string
$06FA	Current command character (initially Control-I)
$077A	Bit 7 = 1 if echo to display is on; bit 6 = 1 if firmware is to generate a LF after CR
$07FA	Current column

Changing port 2 characteristics

Figure 7-3 is a diagram of where the port characteristics are stored and moved under different circumstances. You can see the following from the figure:

- When the power is first turned on, the reset routine moves the predefined startup characteristics listed earlier in this section from ROM into the auxiliary memory screen holes listed in Table 7-9.
- If you specify new characteristics using the system utilities, the utility software changes the values in the auxiliary memory screen holes.
- The values stored in the auxiliary memory screen holes are affected by power-on reset, but not by either Command-Control-Reset or a simple Control-Reset. This feature is provided so that a port that has been reconfigured will remain that way while some other program (such as an application program) is started up.
- IN#2 causes the firmware to move the characteristics stored in the auxiliary memory screen holes into the main-memory screen holes.
- A program can change values in the main-memory screen holes directly. However, the only value guaranteed to be in the same place for the entire Apple II series is the line length in main memory location $057A.
- The firmware uses the port as it is defined in the main-memory screen holes at any given time. You should use the commands listed in Table 7-1 to change these characteristics.

■ **Figure 7-3** Diagram of port 2 characteristics storage



Data format and transmission rate

A noteworthy characteristic of data communication is its strangeness: sometimes the oddest changes make a given communication arrangement work or not work. You must keep this notion firmly in mind when working with a serial port set up as a communications port.

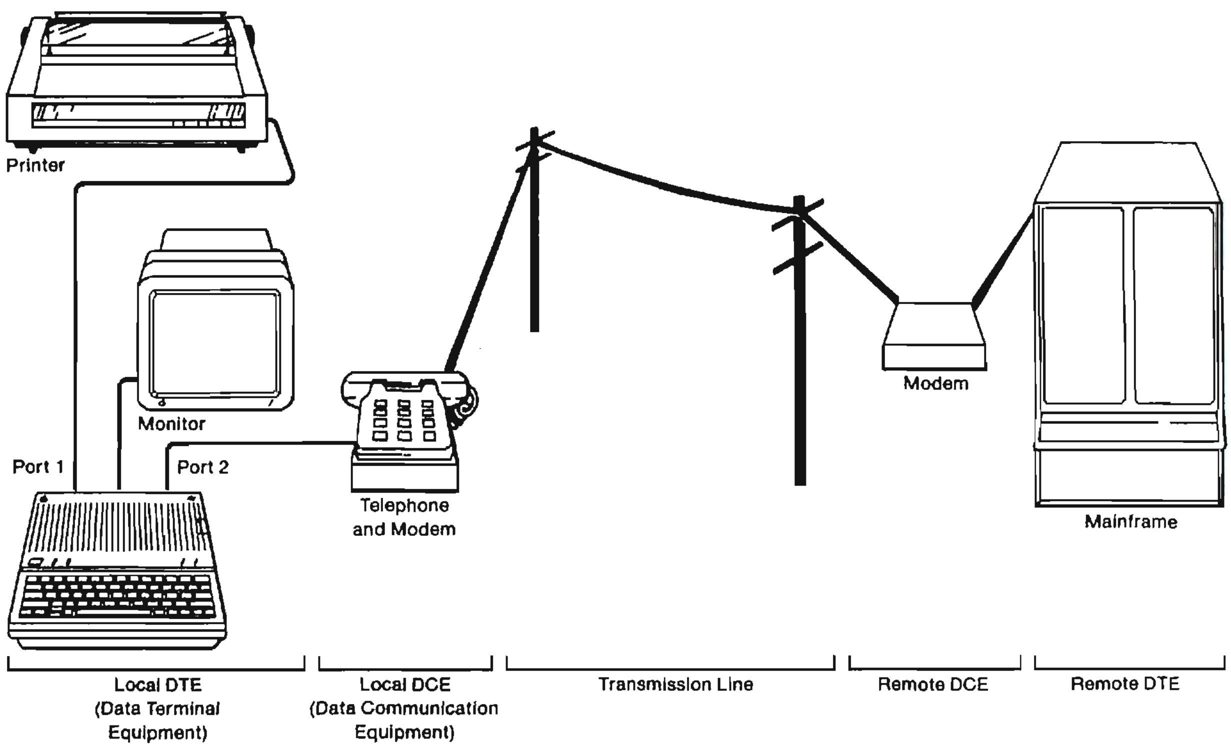
For example, modem communication involves quite a few elements (see Figure 7-4):

- the Apple IIc–family computer and its firmware, with the transmission rate, data format, and other characteristics you have selected
- the cable from the computer to the modem
- the modem
- possibly an acoustic coupler for a telephone handset

- the telephone lines and switching equipment
- some combination of modem, cable, and remote computer or terminal at the distant end

As you can imagine, some care is required for successful data transmission. If you have problems, change only one variable at a time and then cycle through the other variables one at a time. Take nothing for granted. The data format advertised for an information service, for example, may be different from the one you end up using with the Apple IIc–family computer.

■ **Figure 7-4** Devices in a typical communication setup



Carriage return and line feed

If you are communicating with a computer or terminal, carriage return and line feed may or may not be involved. Start off without generating them, and turn on automatic generation only as needed. They are described fully under this same heading in the section on port 1.

Routing input and output

This section discusses the possible ways that serial port 2 can route information. Sometimes the cause of communication problems is that information is not going where you think it is, or it is and you cannot see evidence of the fact. Figures 7-5 through 7-8 show some of the patterns of information flow you can select.

It is best to read all this material as a unit; questions that arise while you read one description may be answered elsewhere.

The simplest serial port 2 command is IN#2 (see Figure 7-5). Port 2 becomes the input device. Data coming into the port gets passed to the input buffer (page $02 of main memory). Applesoft firmware and system software can see the data and carry out commands in the normal way.

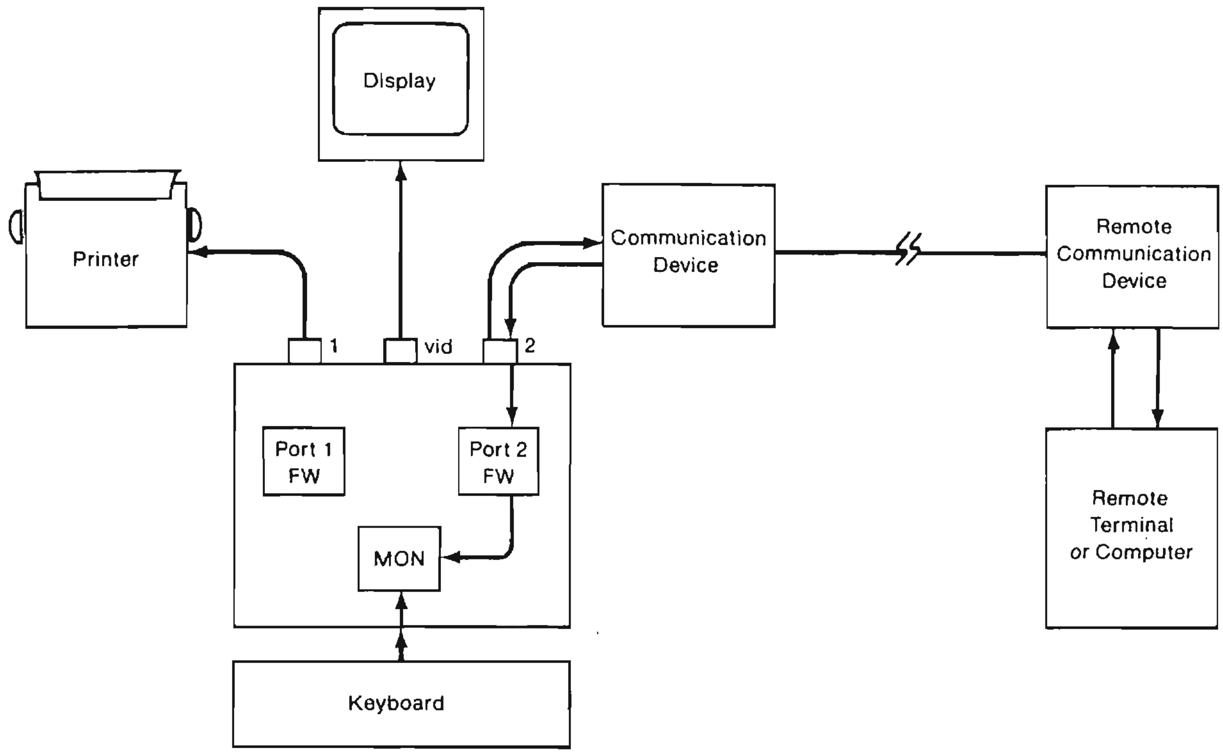
Of course, you can also use just the PR#2 command—for example, if you want to send a listing to the modem.

To use port 2 for data communication, you ordinarily put it into terminal mode. Following IN#2, pressing Control-A gets the attention of the port 2 firmware, which displays a blinking question mark (?) as a prompt. Now type `T` to put the computer in terminal mode. In this mode, the firmware displays a blinking underscore character (_) as a prompt. Terminal mode is further described in the section "Terminal Mode," later in this chapter.

In the discussion that follows, *local* refers to your Apple IIc-family computer. *Remote* refers to some other device, usually in a distant location and at the other end of a communication link. The remote device can be any ASCII-generating unit: a terminal or a computer.

If a remote computer is another Apple IIc-family member or an Apple II-series machine with a Super Serial Card in it, then *most* of the commands described here apply to it as well.

■ **Figure 7-5** Effect of IN#2

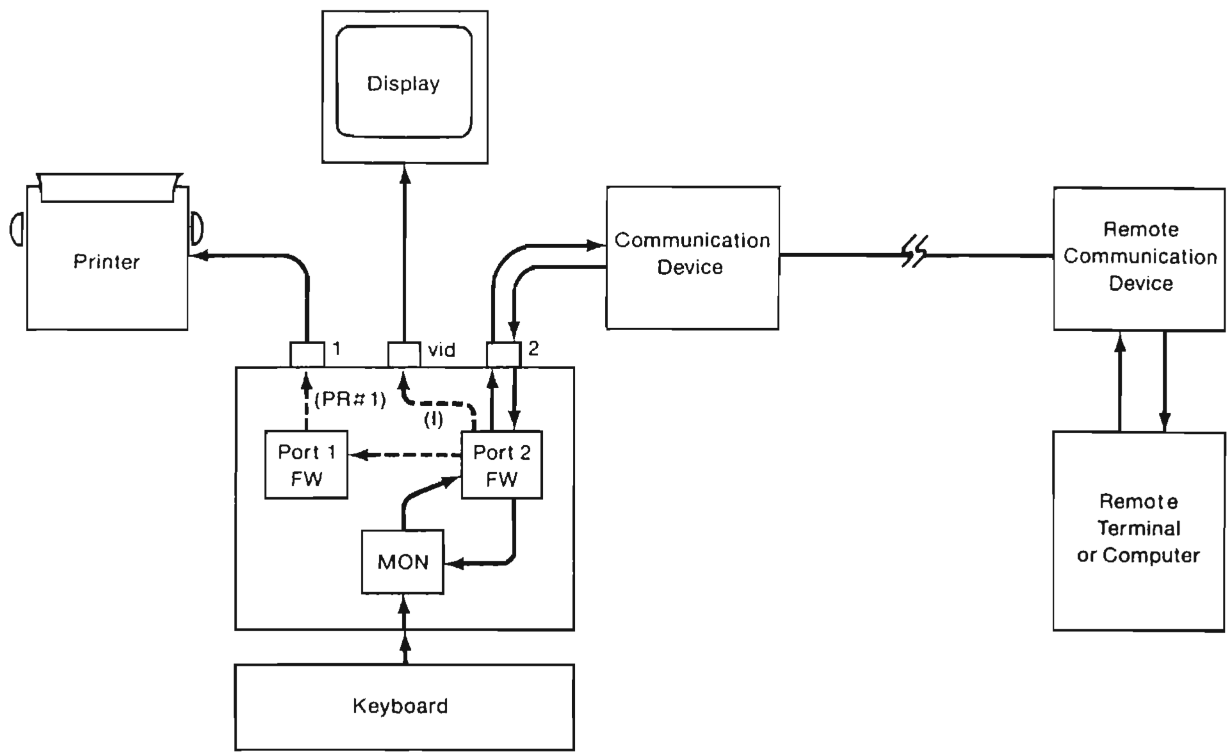


Half-duplex operation

In half-duplex operation, information can flow from A to B or from B to A, but in only one direction at a time. In a half-duplex setup, the host does not echo back to the terminal what the terminal sends it. For half-duplex operation, use IN#2 and Control-A T (Figure 7-6) whether the Apple IIc family member is the host or the terminal.

IN#2 plus Control-A T is the best way to set up the computer for auto-answer operation. The T command allows port 2 firmware to exchange information with the local modem without interference from the local firmware or system software. (The remote device can always cancel the T command with Control-R if necessary, and restore terminal mode with Control-T.) Avoiding PR#2 at this point means that the Apple IIc family member can operate as a half-duplex terminal, half-duplex host, or full-duplex terminal. (The remote device can also issue Control-A PR#2 if PR#2 is required at the local computer.)

■ **Figure 7-6** Effect of IN#2 and T command, half duplex



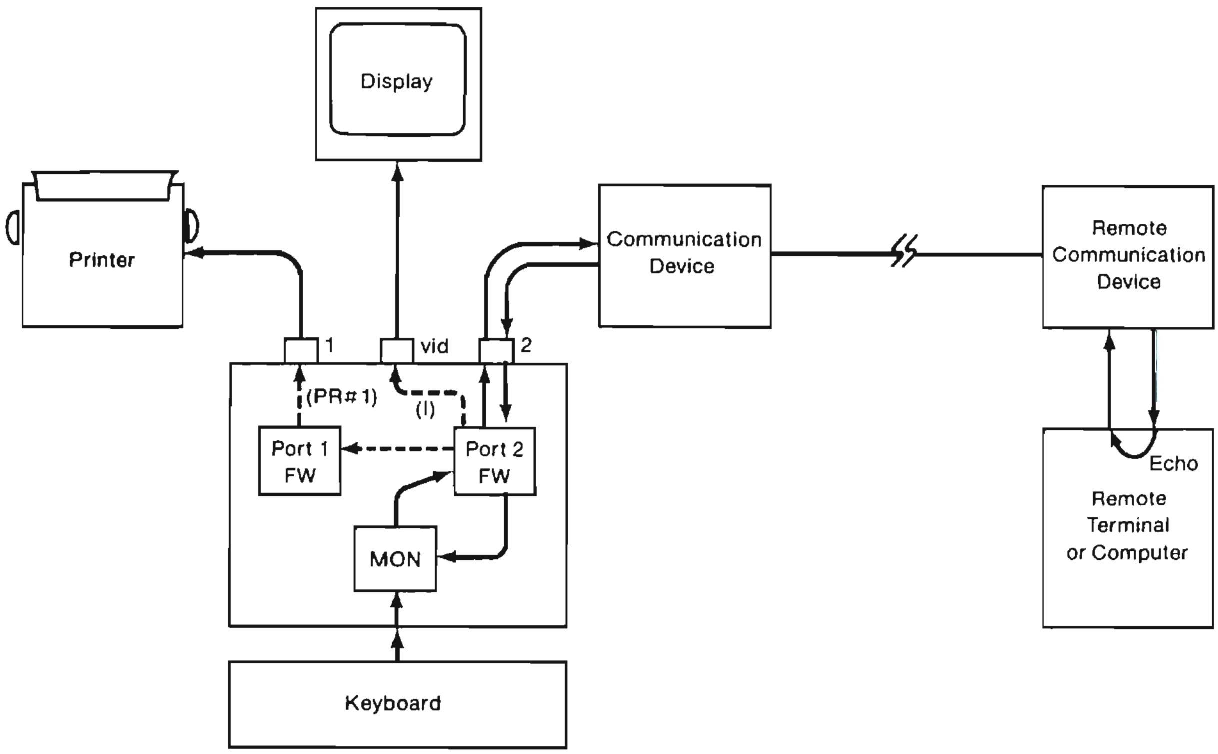
In half-duplex operation, the output hook is available for other uses. For example, you can issue PR#1 to print incoming messages from port 2. Use the Control-A I command to display information on the screen.

Full-duplex operation

In full-duplex operation, information can flow from A to B and from B to A simultaneously. Typically, one of the computers (the host computer) echoes its input to output, so the other computer (the terminal) can easily verify that the communication is taking place.

Figure 7-7 shows the flow of information when the Apple IIc–family computer is a full-duplex terminal. (The setup commands, IN#2 and Control-A T, are the same as for half duplex.)

■ **Figure 7-7** Effect of IN#2 and T command, full-duplex terminal

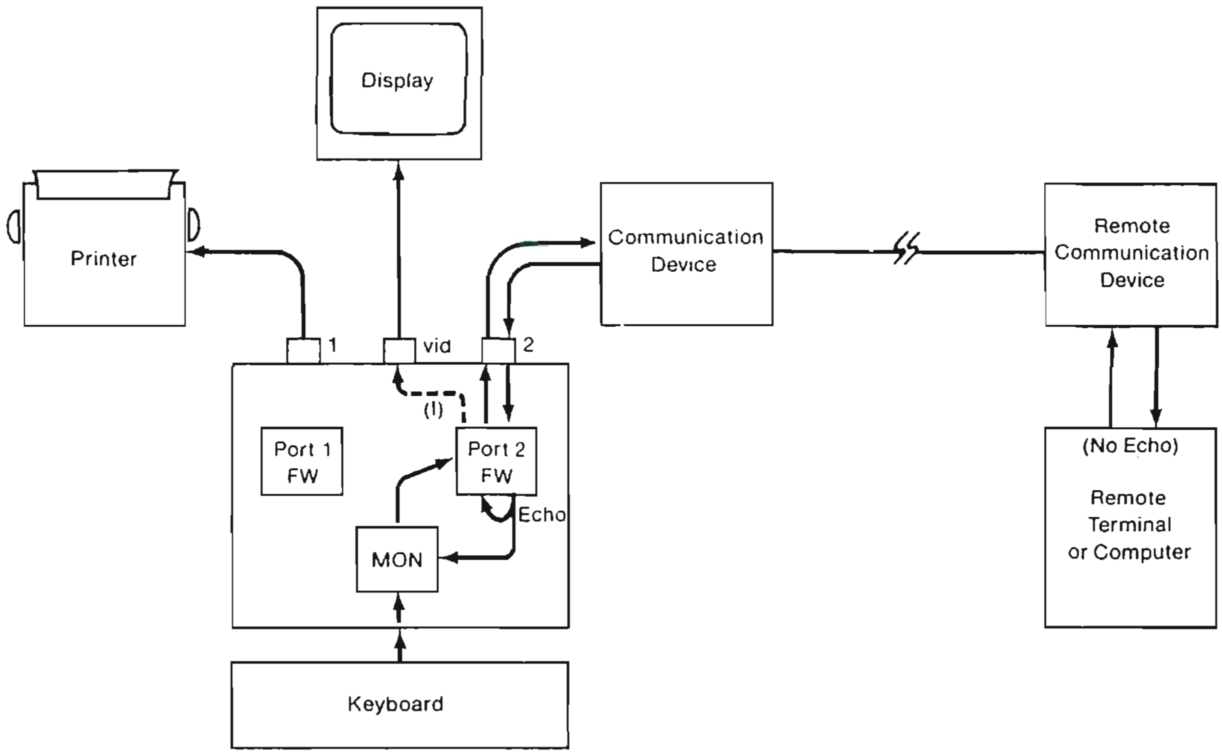


If your Apple IIc–family computer is the terminal in full-duplex operation, use the N command to turn off echoing input to the screen. If the computer does echo input to the screen in this setup, everything you type will appear twice: once from the Apple IIc–family computer and once from the host computer.

In this mode of operation, if you echo input to the printer you can get a printed record of both sides of the communication session: the input from the host, and the Apple IIc–family computer output as echoed by the host.

Figure 7-8 shows the flow of information when the Apple IIc–family computer is a full-duplex host. In this case, the local Apple IIc must echo input to output for the remote device. The setup commands include PR#2 in this case.

■ **Figure 7-8** Effect of IN#2, PR#2, and T command, full-duplex host



▲ **Warning** If the Apple IIc–family computer echoes input to output and the other computer does too, then the first subsequent keypress echoes back and forth endlessly and lock up the Apple IIc–family computer. This requires a Control-Reset to get out.

If you echo input to output when using an information service, the host ends up seeing the echo of what it sent you as though you had typed it. ▲

When the Apple IIc–family computer is a full-duplex host, the local output hook is not available for using the printer or other device. To display keyboard and port 2 input on the screen, issue Control-A I.

Terminal mode

Terminal mode makes the Apple IIc family member act like a dumb terminal—one that just sends and receives information, but does not process it. Input and output flow through special serial I/O buffers on page $08 of auxiliary memory. Applesoft firmware and system software cannot see or interpret the data: only the serial port 2 firmware deals with it.

In most terminal mode setups, the firmware does not display port 2 input unless you use the Control-A I command.

▲ **Warning** When using terminal mode, $0800–$08FF of auxiliary RAM is used for buffering. Any data stored there is overwritten when terminal mode is enabled. ▲

Control-A T turns on terminal mode, and Control-A Q turns it off.

The remote device can go into terminal mode, and then turn off terminal mode at the local Apple IIc with the Control-R command. If it then issues Control-A PR#2, local output will go to the remote device. The remote keyboard and display then become the input and output devices of the local Apple IIc family member's processor. This is called *remote mode*.

In remote mode, the local Apple IIc family member does not use the serial I/O buffers (as it does in terminal mode); therefore, local firmware and system software detect and interpret all input and output data. So, for example, if you type `CATALOG` at the remote device keyboard, the local Apple IIc family member executes the command and lists the disk catalog on the remote device's display. (In terminal mode, the local computer would simply display the word `CATALOG` on its screen.)

The remote device can turn the local Apple IIc's terminal mode back on with Control-T. Control-A T issued at the remote device only turns on the remote device's terminal mode, unless the command character there has already been changed to something else.

Chapter 8 **Block Device I/O**

The Apple IIc family supports both built-in and external block devices. **Block devices** read or write data in groups of bytes called *blocks;* examples of block devices are 5.25-inch disk drives, 3.5-inch disk drives, and memory expansion cards (that is, RAM disks).

Block device I/O is supported by both operating system interface routines and by the SmartPort firmware in the Apple IIc–family ROM. This chapter describes only the interface routines in the Apple IIc and Apple IIc Plus firmware; to learn about the block device interface for a specific operating system, see the reference manual for that operating system. ■

Disk drive I/O

All Apple IIc–family disk drives are block devices, but the various versions of the Apple IIc provide different levels and types of support for them. The following sections describe disk I/O characteristics for each Apple IIc family member. See the section "Disk I/O," in Chapter 11, for descriptions of the disk drive connectors.

Original Apple IIc

In the original Apple IIc, the I/O firmware entry point for the 5.25-inch drives is at $C600 in the ROM. The internal 5.25-inch drive is port 6, drive 1, and the external 5.25-inch drive is port 6, drive 2. The original Apple IIc does not have SmartPort firmware, and thus has no interface for 3.5-inch disk drives.

Table 8-1 summarizes the disk I/O port characteristics of the original Apple IIc.

■ **Table 8-1** Disk port characteristics of the original Apple IIc

Port number	
internal 5.25-inch drive	port 6, drive 1 (valid boot device)
external 5.25-inch drive	port 6, drive 2 (not valid boot device)
Commands	6 Control-K or 6 Control-P from the Monitor
Resets	All resets with a valid reset vector, except Control-Reset, eventually pass control to the built-in disk drive.
Hardware location	$C0E0–C0EF are reserved.
Monitor firmware routines	None.
I/O firmware entry points	$C600 (port 6)
Use of screen holes	Port 6 main- and auxiliary-memory screen holes are reserved.

UniDisk 3.5 and memory expansion Apple IIc

The I/O firmware entry point for the 5.25-inch drives is at $C600 in the main ROM. The internal 5.25-inch drive is port 6, drive 1, and the external 5.25-inch drive is port 6, drive 2.

The I/O firmware entry point for the 3.5-inch disk drives is at $C500 in the main ROM. The first 3.5-inch drive in the chain is port 5, drive 1, and the second is port 5, drive 2.

Table 8-2 summarizes the disk I/O port characteristics for the UniDisk 3.5 and memory expansion versions of the Apple IIc.

■ **Table 8-2** Disk port characteristics of the UniDisk 3.5 and memory expansion Apple IIc

Port number	
internal 5.25-inch drive	port 6, drive 1 (valid boot device)
external 5.25-inch drive	port 6, drive 2 (not valid boot device)
first 3.5-inch drive	port 5, drive 1 (valid boot device)
second 3.5-inch drive	port 5, drive 2 (not valid boot device)
third 3.5-inch drive	port 2, drive 1 (not valid boot device)
Commands	
5.25-inch drives	6 Control-K or 6 Control-P from the Monitor
3.5-inch drives	5 Control-K or 5 Control-P from the Monitor
Resets	All resets with a valid reset vector, except Control-Reset, eventually pass control to port 6, drive 1 or port 5, drive 1, in that order. (If a memory expansion card is installed, control is passed first to the memory expansion card.)
Hardware location	$C0D0–C0DF and $C0E0–C0EF are reserved.
Monitor firmware routines	None.
I/O firmware entry points	$C600 (port 6)
Use of screen holes	Port 6 main- and auxiliary-memory screen holes are reserved

Apple IIc Plus

The I/O firmware entry point for the 5.25-inch drives is at $C600 in the main ROM. The first (external) 5.25-inch drive is port 6, drive 1, and the second drive is port 6, drive 2.

The I/O firmware entry point for the 3.5-inch drives is at $C500 in the main ROM. The internal drive is port 5, drive 1. The first external 3.5-inch drive is port 5, drive 2, the second drive is port 2, drive 1, and the third drive is port 2, drive 2.

Although the last two 3.5-inch drives in the chain are treated as port 2 devices, they are accessed through the $C500 entry point, just as if they were port 5 devices.

Table 8-3 summarizes the disk I/O port characteristics for the Apple IIc Plus.

■ **Table 8-3** Disk port characteristics of the Apple IIc Plus

Port number	
first 5.25-inch drive	port 6, drive 1 (valid boot device)
second 5.25-inch drive	port 6, drive 2 (not valid boot device)
internal 3.5-inch drive	port 5, drive 1 (valid boot device)
first external 3.5-inch drive	port 5, drive 2 (valid boot device)
second external 3.5-inch drive	port 2, drive 1 (not valid boot device)
third external 3.5-inch drive	port 2, drive 2 (not valid boot device)
Commands	
5.25-inch drives	6 Control-K or 6 Control-P from the Monitor
port 5 3.5-inch drives	5 Control-K or 5 Control-P from the Monitor
port 2 3.5-inch drives	5 Control-K or 5 Control-P from the Monitor
Resets	All resets with a valid reset vector, except Control-Reset, eventually pass control to port 5, drive 1; port 5, drive 2; or port 6, drive 1, in that order. (If a memory expansion card is installed, control is passed first to the memory expansion card.)
Hardware location	$C0D0–C0DF and $C0E0–C0EF are reserved.
Monitor firmware routines	None.
I/O firmware entry points	
5.25-inch drives	$C600 (port 6)
port 5 3.5-inch drives	$C500 (port 5)
port 2 3.5-inch drives	$C500 (port 5)
Use of screen holes	Port 6 main- and auxiliary-memory screen holes are reserved

Memory expansion card I/O

A memory expansion card can be plugged into the internal connector in the memory expansion Apple IIc or the Apple IIc Plus. Such a card provides up to 1 MB of RAM, in 256 KB steps, for storage of program and data files. It is not accessed the way main or auxiliary RAM is. Instead, the Apple IIc–family computers treat the memory expansion card the same way they treat a disk drive: as a block device. Thus, you can think of the memory expansion card as a RAM disk; programs can be loaded into the memory expansion card's RAM, but in order to be executed they must be moved, in whole or in part, to the computer's main memory.

The I/O firmware entry point for the memory expansion card is at $C400 in the main ROM of the memory expansion Apple IIc and of the Apple IIc Plus.

The memory expansion card is a block-type device, so I/O operations involving the card use the operating system or the SmartPort I/O interface described later in this chapter.

If there is no memory expansion card installed and you execute a ProDOS ON_LINE call to port 4, ProDOS returns error $2D, indicating a SmartPort BadBlock error.

If there is no memory expansion card installed and you execute a CAT command from BASIC for port 4, BASIC returns the message `PATH NOT FOUND`.

Pascal returns no error when you format the RAM disk, even when the memory expansion card is not installed. However, when the memory expansion card is not installed, a UnitStatus call returns 0 blocks available, and if you try to read the volume in port 4, Pascal returns a value of 8 for IOResult, which indicates that there's no room on the volume. When the memory expansion card is not present, the information for port 4 listed by the Vols command in the Pascal Filer includes `<no dir>` for the volume name and 0 for the number of blocks.

DOS 3.3 does not return any error when you initialize the RAM disk by executing an IN#4 command. However, if you try to read from or write to the RAM disk, DOS 3.3 returns an ONERR error (error code = 8).

You can use a Status call in SmartPort to determine if a memory expansion card is installed. This procedure is discussed in the section "SmartPort Status," later in this chapter.

More information on the Apple memory expansion card can be found in the *Apple IIc Memory Expansion Card Technical Reference*.

△ Original IIc UniDisk 3.5 — The original and UniDisk 3.5 versions of the Apple IIc do not support the memory expansion card. △

△ Apple IIc Plus — The memory expansion card manufactured by Apple Computer, Inc. for use in the memory expansion Apple IIc does not work in the Apple IIc Plus computer. To add a memory expansion card to the Apple IIc Plus, you must purchase a card made by another manufacturer specifically for the Apple IIc Plus. △

Table 8-4 summarizes the I/O port characteristics of the memory expansion card.

■ **Table 8-4** Memory expansion card port characteristics

Port number	port 4 (valid boot device)
Commands	4 Control-K or 4 Control-P from the Monitor
Resets	All resets with a valid reset vector, except Control-Reset, eventually pass control to the memory expansion card.
Hardware location	$C0C0–C0CF are reserved.
Monitor firmware routines	None.
I/O firmware entry points	$C400 (port 4).
Use of screen holes	Port 4 main and auxiliary memory screen holes are reserved.

Using SmartPort

The following sections describe the firmware interface that you can use to access and control block devices in your programs. This interface is present in the Apple IIc Plus and in all versions of the Apple IIc *except* the original Apple IIc.

◆ *Protocol Converter:* The SmartPort and the Protocol Converter are essentially the same firmware interface with different names. All the specifications given in this manual for the SmartPort apply to the Protocol Converter as well.

Locating SmartPort

Your SmartPort routines should always begin with a search for the following signature bytes:

$Cn01=$20

$Cn03=$00

$Cn05=$03

$Cn07=$00

The port or slot number, *n*, can be an integer from 1 to 7. The SmartPort entry point (the dispatch routine) is then found at address $Cn00 + ($CnFF) + 3, where *($CnFF)* refers to the value of the byte located at $CnFF. The sample program at the end of this chapter illustrates such a search.

There is a SmartPort ID Type byte (Table 8-5) located at $CnFB that indicates which type of device is supported by the SmartPort firmware for that port or slot. For example, the SmartPort ID Type byte for port 4 in the Apple IIc Plus computer is $01, indicating SmartPort support for a RAM card (a memory expansion card). The ID Type byte for port 5 of all Apple IIc–family computers (except for the original Apple IIc, which does not have SmartPort) is $00, indicating SmartPort support for disk drives.

■ **Table 8-5** SmartPort ID Type byte

Bit	Meaning when bit set
7	Extended SmartPort calls
6	Reserved
5	Reserved
4	Reserved
3	Reserved
2	Reserved
1	SCSI devices
0	RAM cards

△ Important The SmartPort firmware is present in port 4 of the memory expansion Apple IIc and of the Apple IIc Plus, even when the memory expansion card is not installed. To check for the memory expansion card, issue a Status call, status code $03, to the port 4 SmartPort. If the data returned indicates 0 bytes available, the card is not present. △

Issuing a call to SmartPort

SmartPort calls are coded like ProDOS Machine Language Interface (MLI) calls: the program executes a JSR to the SmartPort dispatch routine for the appropriate port. For example, to access the memory expansion card in the Apple IIc Plus, your program would execute a JSR to address $C400 + ($C4FF) + 3, where ($C4FF) refers to the value of the byte located at $C4FF.

The SmartPort command number and a 2-byte pointer to the call's parameter list must immediately follow the call. Here is an example of a call to the SmartPort firmware:

```
SPCALL    JSR     DISPATCH    ;Jump to SmartPort command dispatcher entry point
          DFB     CmdNum      ;Specifies the command type
          DW      CmdList     ;2-byte (low, high) pointer to parameter list
          BCS     ERROR       ;Sets carry on an error
```

The command number (CmdNum) defines which SmartPort call you want to make. All SmartPort calls include a 2-byte pointer to a parameter list. The parameter list can contain information to be used by the call, or can provide space for information to be returned by the call. The length and content of the parameter list depend on the call being made. The format of each SmartPort call's parameter list is described later in this chapter.

▲ **Warning** You must leave 35 bytes of stack space available to the block device interface (SmartPort or operating system).

All RAM that is accessed by the interface must be both read and write enabled. See the section "Bank-Switched Memory," in Chapter 2, for the commands to read and write enable RAM.

You must not pass any data to or from the SmartPort through any zero page locations. SmartPort uses zero page for temporary storage of data, and might corrupt the data you stored there.

If you fail to observe these cautions, your program will crash. ▲

When the call has finished, the program resumes execution at the statement following the pointer to the parameter list. In the example above, the DFB and DW statements are skipped and execution resumes with the BCS statement. If the call is successful, the c flag (in the Processor Status register) is cleared (0), and the A register (accumulator) is cleared to all 0's. If the call is unsuccessful, the c flag is set (1) and the error code is placed in the A register. After the SmartPort call, the contents of the 65C02's registers are as shown in Table 8-6.

■ **Table 8-6** Contents of the 65C02 registers after a SmartPort call

Register	Processor Status								X	Y	A	PC	S
Bit	n	v	1	b	d	i	z	c					
Successful call	x	x	1	x	0	u	x	0	n	n	0	JSR+3	u
Unsuccessful call	x	x	1	x	0	u	x	1	x	x	Error	JSR+3	u

x = undefined, except in cases where index information is returned in X and Y registers
u = unchanged
n = number of bytes transferred when the transfer was from the device to the host; otherwise undefined

▲ **Warning** The extended SmartPort calls available for use on the Apple IIGS computer cannot be used on the Apple IIc–family computers. Using an extended SmartPort call causes unpredictable results on an Apple IIc or Apple IIc Plus. ▲

SmartPort assignment of unit numbers

The unit number specifies which device is to respond to the SmartPort command. The unit number is included in every parameter list. Calls that allow you to reference the SmartPort itself use a unit number of $00. Only Status, Init, and Control calls may be made to unit $00.

The UniDisk 3.5 Apple IIc has SmartPort support only for port 5. The memory expansion Apple IIc and the Apple IIc Plus have SmartPort support for ports 5 (disk drives) and 4 (memory expansion card). SmartPort assigns unit numbers to devices in ascending order starting with unit number $01. The highest unit number supported by port 5 is $06; however, due to power-supply limitations, $04 is the highest number that should ever be used. For port 4, only unit number $01 is assigned.

In the UniDisk 3.5 and memory expansion versions of the Apple IIc, the UniDisk 3.5 drive closest to the computer is assigned unit number $01; additional UniDisk 3.5 drives (if any) are assigned unit numbers according to their positions in the daisy chain. In the Apple IIc Plus, the built-in disk drive is assigned unit number $01 and additional 3.5-inch disk drives are assigned unit numbers from $02 to $04, according to their positions in the chain. No unit numbers are assigned to 5.25-inch disk drives.

◆ *Note:* ProDOS 1.1.1 supports only two devices per slot. ProDOS 1.2 and later supports up to four devices on SmartPort. ProDOS 1.2 maps devices $03 and $04 so that they appear to software as if they were connected to port 2. Any external 5.25-inch disk drives must come last in the daisy chain, and are mapped to port 6.

External disk drive options

The original Apple IIc has one internal 5.25-inch disk drive and supports one external 5.25-inch drive. Only the internal 5.25-inch disk drive can be used as a startup device for a cold start.

The UniDisk 3.5 Apple IIc has one internal 5.25-inch disk drive, and supports one external 5.25-inch drive and one external UniDisk 3.5 drive. If both an external 5.25-inch disk drive and a UniDisk 3.5 drive are used, the 5.25-inch disk drive must come after the UniDisk 3.5 drive; that is, the UniDisk 3.5 drive is attached directly to the computer, and the 5.25-inch drive is connected to the UniDisk 3.5.

The possible startup devices for the UniDisk 3.5 Apple IIc are, in order of priority, as follows:

- internal 5.25-inch drive at $C600 (port 6, drive 1)
- external 3.5-inch drive at $C500 (port 5, drive 1)

The memory expansion Apple IIc has one internal 5.25-inch disk drive, and supports up to three external devices plus the memory expansion card. The external devices can be any combination of up to three external UniDisk 3.5 drives and one external 5.25-inch drive. If an external 5.25-inch drive is used, it must be the last device in the daisy chain.

The possible startup devices for the memory expansion Apple IIc are, in order of priority, as follows:

- memory expansion card at $C400 (port 4, drive 1)
- internal 5.25-inch drive at $C600 (port 6, drive 1)
- external UniDisk 3.5 drive at $C500 (port 5, drive 1)

The Apple IIc Plus has one internal 3.5-inch disk drive, and supports up to three external drives plus the memory expansion card. The three external drives can be any combination of up to two 5.25-inch drives, up to three UniDisk 3.5 drives, and one Apple 3.5 drive. An Apple 3.5 drive must come before any UniDisk 3.5 drives in the daisy chain, and any 5.25-inch drives must come last, after any 3.5-inch drives in the daisy chain.

The possible startup devices for Apple IIc Plus are, in order of priority, as follows:

- memory expansion card at $C400 (port 4, drive 1)
- internal 3.5-inch drive at $C500 (port 5, drive 1)
- external 3.5-inch drive at $C500 (port 5, drive 2)
- external 5.25-inch drive at $C600 (port 6, drive 1)

Booting from 5.25-inch disk drives is handled by the slot 6 firmware.

Reading the SmartPort call descriptions

The SmartPort call descriptions in this chapter include the following items:

Command name	The name used to identify the SmartPort call. The command name is given as the heading to the SmartPort call description.
Command number	A 1-byte number that specifies the type of SmartPort call. The command number is contiguous in memory with the JSR instruction to the SmartPort entry point. The command number is included in the heading to the SmartPort call description.

Parameter list — A list of the required call parameters. Each SmartPort call includes a 2-byte pointer to the parameter list. The parameter-list pointer is contiguous in memory with the command number.

Possible errors — A list of the error codes that can be returned by the call. The possible causes of each error are discussed in the section "Summary of SmartPort Error Codes," at the end of this chapter.

The parameter list descriptions in this chapter include the following items. Not all parameter types are used by every SmartPort call.

Parameter count — A 1-byte number that specifies the number of parameters in the parameter list. The parameter count is the first item in the parameter list.

Unit number — A 1-byte value that specifies the device to or from which the SmartPort call is to direct I/O. The unit number is the second item in the parameter list.

Buffer address — A 2-byte (word) pointer to the start of the block of memory that SmartPort is to use in the I/O transaction. The buffer address is referred to in the parameter description by a name that reflects the use of the buffer; for example, *Status list* or *Control list*.

Block number — A 3-byte number specifying the logical address in a block device of the block of data to be used in an I/O transaction. The block number is translated into a physical address by the controller in the block device.

Byte count — A 2-byte number specifying the number of bytes to be transferred between memory and the device.

Address pointer — A 3-byte number specifying an address within the device. The meaning of this number depends on the specific device being accessed.

Generic SmartPort calls

Generic SmartPort calls are commands that can be used with any device attached to SmartPort, as opposed to device-specific SmartPort calls. Generic SmartPort calls are described in detail in the following sections. Device-specific SmartPort calls are described later in this chapter.

Status ($00)

Description The Status call returns information about a particular device or about the SmartPort itself. Only Status calls that return general information are listed here. Some devices recognize other Status calls that provide diagnostic or other information specific to that device. Device-specific Status calls must be implemented with a status code of $04 or greater. On return from a Status call, the X and Y registers contain a count of the number of bytes transferred to the host. The X register contains the low byte of the count, and the Y register contains the high byte of the count.

Parameter list
Parameter count
Unit number
Status list pointer (low byte)
Status list pointer (high byte)
Status code

Parameter descriptions

Parameter count: Byte value = $03

Unit number: 1-byte value in the range $00 to $7E

SmartPort assigns a unique number to each device at initialization time, as described in the section "SmartPort Assignment of Unit Numbers," earlier in this chapter. A Status call with a unit number of $00 and a status code of $00 returns the status of the SmartPort itself, as described under "SmartPort Status," later in this call description.

Status list pointer: 2-byte pointer

This parameter is a pointer to the buffer to which the status list is to be returned. The length of the buffer varies, depending on the status request being made.

Status code: 1-byte value in the range $00 to $FF

This number indicates the kind of status request being made. All devices respond to the following requests:

Status code	*Status returned*
$00	Return device status
$01	Return device control block
$02	Return newline status (character devices only)
$03	Return device information block

Although devices must respond to the preceding status requests, a device may not be able to support the request. In this case, the device returns an invalid status code error ($21).

Device-specific status codes are described in the section "Device-Specific SmartPort Calls," later in this chapter.

Status code = $00

If the status code is $00, the device returns the device status block (DSB). The DSB consists of 4 bytes. The first is the general status byte:

Bit	Function
7	1 = block device; 0 = character device
6	1 = write allowed
5	1 = read allowed
4	1 = device on line or disk in drive
3	1 = format allowed
2	1 = media write-protected (block devices only)
1	Reserved; must = 0
0	1 = device currently open (character devices only)

If the device is a block device, the next 3 bytes indicate the number of blocks in the device. The least significant byte is first. If the device is a character device, these bytes are set to 0.

◆ *Note:* There are currently no character devices available for use on SmartPort.

Status code = $01

If the status code is $01, the device returns the device control block (DCB). The size and contents of the DCB are device dependent. The DCB is typically used to control various operating characteristics of a device. Each device has a default DCB, which you can alter with a Control call. The first byte of the DCB (the *count byte*) indicates the number of bytes in the DCB, excluding the count byte. A count-byte value of $00 indicates a DCB length of 256. The length of the DCB is always in the range 1 to 256 bytes, excluding the count byte.

△ **UniDisk 3.5** The UniDisk 3.5 disk drive has no DCB, and returns an error (BadCtl $21) in response to a Status call with status code equal to $01. △

Status code = $02

No character devices are currently implemented for use on the SmartPort, so the newline status is presently undefined.

Status code = $03

If the status code is $03, the device returns the device information block (DIB). The DIB contains information identifying the device, indicating the type of device, and specifying various other attributes. The returned status list has the following form:

Byte	Function
1	Device status (1 byte)
2	Block size (low byte)
3	Block size (middle byte)
4	Block size (high byte)
5	ID string length in bytes (1 byte, $10 is maximum value)
6–21	ID string (16 bytes)
22	Device type (1 byte)
23	Device subtype (1 byte)
24–25	Firmware version (2 bytes)

The uses of these fields are as follows:

- The device status byte is the same as the general status byte returned in the DSB (status code = $00).
- The block size field is the same as the block size field returned in the DSB.
- The ID string consists of a count byte indicating the number of ASCII characters in the ID string, followed by a 16-byte field containing an ASCII string identifying the device. The most significant bit of each ASCII character is set to 0. If the ASCII string consists of fewer than 16 characters, ASCII space characters ($20) are used to fill the unused portion of the string.

- The device type and device subtype fields are 1-byte fields. Certain devices have been assigned types and subtypes, as follows:

Type	Subtype	Device
$00	$00	Apple II memory expansion card
$01	$00	UniDisk 3.5
$01	$40	Apple 3.5 drive
$02		ProFile™ hard disk
$03–$09		Reserved
$0A		5.25-inch disk
$0B–$0F		Reserved

△ Apple IIc Plus In some cases, the Apple IIc Plus computer might return a subtype of $00 for an Apple 3.5 drive. To determine whether a device with a type of $01 and a subtype of $00 is a UniDisk 3.5 or an Apple 3.5 disk drive, issue a Status call with a status code of $05 to the device. This device-specific status call returns the UniDisk status if the drive is a UniDisk 3.5. If the device is an Apple 3.5 disk drive, it will return an invalid status code error ($21). △

The 3 most significant bits in the DIB device-subtype byte indicate whether a device supports the extended SmartPort interface, disk-switched errors, or removable media, as follows:

Bit	Function
7	1 = supports extended SmartPort interface
6	1 = provides error if disk has been switched
5	0 = removable media
4–0	Reserved

◆ *Note:* The Apple IIc memory expansion card does not follow the subtype convention in that the memory expansion card has a subtype of $00, but does not have removable media.

- The firmware version is a 2-byte number indicating the version of the firmware in the device.

SmartPort status

A Status call with a unit number of $00 and a status code of $00 is a request to return the status of the SmartPort driver. The status list is 8 bytes long. The first byte indicates the total number of devices connected to the port; the other 7 bytes are reserved.

For port 4 in the Apple IIc Plus and in the memory expansion Apple IIc with the ROM revision (ID byte $FBBF is $04), the first byte of the SmartPort status list is $00 if no memory expansion card is installed and $01 if a memory expansion card is present. In the memory expansion Apple IIc without the ROM revision (ID byte $FBBF is $03), the first byte of the SmartPort status list is $01 whether the memory expansion card is installed or not.

Possible errors

The following error values can be returned by the Status call:

$06	BusErr	Communications error
$21	BadCtl	Invalid status code
$30–$3F; $50–$7F		Device-specific error

ReadBlock ($01)

Description The ReadBlock call reads one 512-byte block from the block device specified by the unit number. The block is read into memory starting at the address specified by the data buffer pointer passed in the parameter list.

Parameter list

- Parameter count
- Unit number
- Data buffer pointer (low byte)
- Data buffer pointer (high byte)
- Block number (low byte)
- Block number (middle byte)
- Block number (high byte)

Parameter descriptions

Parameter count: Byte value = $03

Unit number: 1-byte value in the range $01 to $7E

SmartPort assigns a unique number to each device at initialization time, as described in the section "SmartPort Assignment of Unit Numbers," earlier in this chapter.

Data buffer pointer: 2-byte pointer

This parameter is a pointer to the beginning of the buffer into which the data is to be read. The buffer must be 512 bytes long.

Block number: 3-byte number

The block number is the logical address in the device of a block of data to be read. There is no general connection between block numbers and the layout of tracks and sectors on the disk. Translation from logical to physical address is performed by the device.

Possible errors The following error values can be returned by the ReadBlock call:

$06	BusErr	Communications error
$27	IOError	I/O error
$28	NoDrive	No device connected
$2D	BadBlock	Invalid block number
$2F	OffLine	Device off line or no disk in drive

WriteBlock ($02)

Description The WriteBlock call writes one 512-byte block to the block device specified by the unit number. The block is written from memory starting at the address specified by the data buffer pointer passed in the parameter list.

Parameter list

Parameter count
Unit number
Data buffer pointer (low byte)
Data buffer pointer (high byte)
Block number (low byte)
Block number (middle byte)
Block number (high byte)

Parameter descriptions

Parameter count: Byte value = $03

Unit number: 1-byte value in the range $01 to $7E

SmartPort assigns a unique number to each device at initialization time, as described in the section "SmartPort Assignment of Unit Numbers," earlier in this chapter.

Data buffer pointer: 2-byte pointer

The data buffer pointer points to a buffer in memory that contains the data that is to be written to the device. The buffer must be 512 bytes long.

Block number: 3-byte number

The block number is the logical address in the device of a block of data to be written. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical address is performed by the device.

Possible errors The following error values can be returned by the WriteBlock call:

$06	BusErr	Communications error
$27	IOError	I/O error
$28	NoDrive	No device connected
$2B	NoWrite	Disk write-protected
$2D	BadBlock	Invalid block number
$2F	OffLine	Device off line or no disk in drive

Format ($03)

Description The Format call formats a block device. The formatting performed by this call is not linked to any operating system; it only prepares all blocks on the medium for reading and writing. Operating-system-specific catalog information, such as bit maps and directories, are not prepared by this call.

Parameter list Parameter count
Unit number

Implementation of the Format call Some block devices may require device-specific information at format time. This device-specific information may include a list of bad blocks (referred to as a *spare list*) to be written following physical formatting of the medium. In this case, it may not be desirable to implement the Format call so that a physical format is actually performed because the spare list might exist only in the device (that is, a backup spare list may not be available from the vendor), or because of the time involved in executing a bad-block scan. It may be more desirable to implement device-specific Control calls to lay down the physical tracks and to initialize the spare list. If this latter procedure is followed, the device driver need only return to the application with the A register set to $00 and the carry flag cleared in response to a format request. This procedure should be used only when it is not desirable to physically format the medium.

Parameter descriptions

Parameter count: Byte value = $01

Unit number: Byte value in the range $01 to $7E

SmartPort assigns a unique number to each device at initialization time, as described in the section "SmartPort Assignment of Unit Numbers," earlier in this chapter.

Possible errors The following error values can be returned by the Format call:

$06	BusErr	Communications error
$27	IOError	I/O error
$28	NoDrive	No device connected
$2B	NoWrite	Disk write-protected
$2F	OffLine	Device off line or no disk in drive

Control ($04)

Description The Control call sends control information to the device. The information may be either general or device specific.

Parameter list

Parameter count
Unit number
Control list pointer (low byte)
Control list pointer (high byte)
Control code

Parameter descriptions

Parameter count: Byte value = $03

Unit number: 1-byte value in the range $00 to $7E

SmartPort assigns a unique number to each device at initialization time, as described in the section "SmartPort Assignment of Unit Numbers," earlier in this chapter.

Control list pointer: Word pointer (bank $00)

This parameter is a pointer to the buffer from which the control information is to be read. The first 2 bytes of the control list (the count bytes, low byte first) specify the length of the control list (not including the count bytes). A control list is mandatory, even if the call being issued does not pass information in the list. In this latter case, the count bytes are equal to $00.

Control code: 1-byte value in the range \$00 to \$FF

The control code is the number of the control request being made. This number and the function indicated are device specific, except that all devices must reserve the following codes for the specific functions indicated:

Control code	Function
\$00	Resets the device
\$01	Sets device control block

Control code = \$00

If the control code is \$00, the Control call performs a software reset of the device. It generally returns control parameters values to default values.

△ **UniDisk 3.5** The UniDisk 3.5 disk drive does not respond to the Reset call. See the description of the Execute extension to the Control call in the section "SmartPort Calls Specific to UniDisk 3.5," later in this chapter, for a way to reset the UniDisk 3.5. △

Control code = \$01

If the control code is \$01, the Control call alters the contents of the device control block (DCB). Devices generally use the bytes in this block to control global aspects of the device's operating environment. Because the length of the DCB is device dependent, the recommended way to set the DCB is to read in the DCB (with the Status call), alter the bits of interest, and then write the same string with the Control call. The first byte is the length of the DCB, excluding the byte itself. A value of \$00 for the length byte indicates a DCB size of 256 bytes.

△ **UniDisk 3.5** The UniDisk 3.5 disk drive has no DCB, and returns an error (BadCtl \$21) in response to a Control call with control code equal to \$01. △

Possible errors The following error values can be returned by the Control call:

$06	BusErr	Communications error
$21	BadCtl	Invalid control code
$22	BadCtlParm	Invalid parameter list
$30–$3F	Undefined	Device-specific error

Init ($05)

Description The Init call causes SmartPort to execute its initialization sequence, causing a hardware reset of all devices and assigning a unit number to each device. This call is not made to a specific unit; rather, it is made to the SmartPort as a whole. This call is executed automatically on system startup or forced cold reset.

△ **Important** The Init call must not be executed by an application designed to run on the Apple IIGS. Issuing this call on the Apple IIGS may assign unit numbers to devices different from those in the ProDOS device list. Applications wishing to reset a sepcific device should use the Control call with a control code of $00. △

Parameter list Parameter count
Unit number

Parameter descriptions

Parameter count: Byte value = $01

Unit number: Byte value = $00

The unit number used in this call is always $00. This call must be made to the SmartPort itself; it cannot be made to an individual device.

Possible errors The following error values can be returned by the SmartPort Init call:

$06	BusErr	Communications error
$28	NoDrive	No device connected

Open ($06)

Description The Open call prepares a character device for reading or writing.

◆ *Note:* A block device does not accept this call, but returns an invalid command error ($01). There are currently no character devices implemented for use on SmartPort.

Parameter list Parameter count
Unit number

Parameter descriptions

Parameter count: Byte value = $01

Unit number: 1-byte value in the range $01 to $7E

SmartPort assigns a unique number to each device at initialization time, as described in the section "SmartPort Assignment of Unit Numbers," earlier in this chapter.

Possible errors The following error values can be returned by the Open call:

$01	BadCmd	Invalid command
$06	BusErr	Communications error
$28	NoDrive	No device connected

Close ($07)

Description The Close call tells a character device that a sequence of read or write operations has ended. For a printer, this call could have the effect of flushing the print buffer.

◆ *Note:* A block device does not accept this call, but returns an invalid command error ($01). There are currently no character devices implemented for use on SmartPort.

Parameter list Parameter count
Unit number

Parameter descriptions

Parameter count: Byte value = $01

Unit number: 1-byte value in the range $01 to $7E

SmartPort assigns a unique number to each device at initialization time, as described in the section "SmartPort Assignment of Unit Numbers," earlier in this chapter.

Possible errors The following error values can be returned by the Close call:

$01	BadCmd	Invalid command
$06	BusErr	Communications error
$28	NoDrive	No device connected

Read ($08)

Description The Read call reads into memory the number of bytes specified by the byte count parameter. Although this call is generally intended for use by character devices, the Read call can be used to read a block of nonstandard size from a block device. In this latter case, the address pointer parameter is interpreted as a block address.

Parameter list

Parameter count
Unit number
Data buffer pointer (low byte)
Data buffer pointer (high byte)
Byte count (low byte)
Byte count (high byte)
Address pointer (low byte)
Address pointer (middle byte)
Address pointer (high byte)

Parameter descriptions

Parameter count: Byte value = $04

Unit number: 1-byte value in the range $01 to $7E

SmartPort assigns a unique number to each device at initialization time, as described in the section "SmartPort Assignment of Unit Numbers," earlier in this chapter.

Data buffer pointer: 2-byte pointer

This parameter is a pointer to a buffer into which the data is to be read. The buffer must be large enough to accommodate the number of bytes requested.

Byte count: 2-byte number

The byte count specifies the number of bytes to be transferred. Specific implementations of the SmartPort using the SmartPort Bus have limitations on this number; for example, for the UniDisk 3.5 you cannot transfer more than 767 bytes. For the memory expansion card and the Apple 3.5 disk drive, the byte count is limited to 512 bytes.

△ Apple IIc Plus In the Apple IIc Plus computer, 512 bytes are transferred no matter what value you enter for the byte count. △

Address pointer: 3-byte address

The address pointer specifies the address within the device from which the bytes are to be read. The meaning of the address parameter depends on the device involved. This parameter can be used with a block device to specify a block address in order to read a block of nonstandard size. For example, the use of this parameter with the Read call makes it possible for an Apple 3.5 drive or UniDisk 3.5 drive to read 524-byte Macintosh blocks from 3.5-inch disks.

Possible errors The following error values can be returned by the Read call:

$06	BusErr	Communications error
$27	IOError	I/O error
$28	NoDrive	No device connected
$2B	NoWrite	Disk write protected
$2D	BadBlock	Invalid block number
$2F	OffLine	Device off line or no disk in drive

Write ($09)

Description The Write call writes the number of bytes specified by the byte count to the device specified by the unit number. Although this call is generally intended for use by character devices, the Write call can be used to write a block of nonstandard size to a block device. In this latter case, the address pointer parameter is interpreted as a block address.

Parameter list

Parameter count
Unit number
Data buffer pointer (low byte)
Data buffer pointer (high byte)
Byte count (low byte)
Byte count (high byte)
Address pointer (low byte)
Address pointer (middle byte)
Address pointer (high byte)

Parameter descriptions

Parameter count: Byte value = $04

Unit number: 1-byte value in the range $01 to $7E

SmartPort assigns a unique number to each device at initialization time, as described in the section "SmartPort Assignment of Unit Numbers," earlier in this chapter.

Data buffer pointer: 2-byte pointer

This parameter is a pointer to a buffer that contains the data to be written to the device. The buffer must be large enough to accommodate the number of bytes specified.

Byte count: 2-byte number

The byte count specifies the number of bytes to be transferred. All of the current implementations of the SmartPort using the SmartPort Bus have a limitation of 767 bytes. Other peripheral cards supporting the SmartPort interface may not have this limitation.

Address pointer: 3-byte address

The address pointer specifies the address within the device to which the bytes are to be written. The meaning of the address parameter depends on the device involved. This parameter can be used with a block device to specify a block address in order to write a block of nonstandard size. For example, the use of this parameter with the Write call makes it possible for an Apple 3.5 drive or UniDisk 3.5 drive to write 524-byte Macintosh blocks to 3.5-inch disks.

Possible errors The following error values can be returned by the Write call:

\$06	BusErr	Communications error
\$27	IOError	I/O error
\$28	NoDrive	No device connected
\$2B	NoWrite	Disk write protected
\$2D	BadBlock	Invalid block number
\$2F	OffLine	Device off line or no disk in drive

Device-specific SmartPort calls

In addition to the common command set of SmartPort calls already listed, a device may recognize its own device-specific calls. Usually, these calls are implemented as a subset of the SmartPort Status or Control calls rather than as new commands. The following sections describe device-specific SmartPort calls for Apple 3.5 disk drives and UniDisk 3.5 disk drives.

SmartPort calls specific to the Apple 3.5 disk drive

One device-specific call is provided as an extension to the Control call for the Apple 3.5 drive. This device-specific control call may be used only with the Apple 3.5 drive. To determine whether a device is an Apple 3.5 drive, address a Status call with a status code of $03 to the device. If it is an Apple 3.5 drive, the device returns a type of $01 and a subtype of either $00 or $40. If it returns a subtype of $00, address a Status call with a status code of $05 to the device. If it returns an invalid status code error ($21), then it is an Apple 3.5 disk drive.

Because this device-specific call to the Apple 3.5 drive is implemented as an extension to the Control call, only the control code and control list for this call are described here. Refer to the section on the Control call earlier in this chapter for information about the command byte and parameter list.

The Apple 3.5 disk drive extension to the Control call is as follows:

Control code	Name	Function
$04	Eject	Ejects a disk

◆ *Note:* There are several device-specific SmartPort calls for the Apple 3.5 disk drive available on the Apple IIGS in addition to the one implemented on the Apple IIc Plus. For descriptions of these calls, see the chapter on SmartPort in the *Apple IIGS Firmware Reference.*

Eject

The Eject extension to the Control call ejects a disk from the disk drive.

Control code

The control code for the Apple 3.5 disk drive Eject extension to the Control call is $04.

Control list

The control list for this call has a length of $0000.

SmartPort calls specific to UniDisk 3.5

Five UniDisk 3.5 device-specific calls are provided as extensions to the Control and Status calls. These device-specific calls may be used only with the UniDisk 3.5. To determine whether a device is a UniDisk 3.5 drive, address a Status call with a status code of $03 to the device. If it is a UniDisk 3.5 drive, the device returns a type of $01 and a subtype of $00.

△ **Apple IIc Plus** In some cases, the Apple IIc Plus computer may return a type of $01 and a subtype of $00 for an Apple 3.5 drive. To determine whether a device with a type of $01 and a subtype of $00 is a UniDisk 3.5 or an Apple 3.5 disk drive, issue a Status call with a status code of $05 to the device. If the device is an Apple 3.5 disk drive, it will return an invalid status code error ($21). △

For calls implemented as extensions to the Control call, only the control code and control list are described in this section. For calls implemented as extensions to the Status call, only the status code and status list are described. Refer to the sections on the Control and Status calls earlier in this chapter for more information about these calls.

The UniDisk 3.5 disk drive extensions to the Control call are as follows:

Control code	Name	Function
$04	Eject	Ejects a disk
$05	Execute	Executes a 65C02 routine
$06	Self Address	Sets an address in the UniDisk 3.5 to which to download a 65C02 routine
$07	Download	Downloads a 65C02 routine into the UniDisk 3.5

The UniDisk 3.5 disk drive extension to the Status call is as follows:

Status code	Name	Status returned
$05	UniDiskStat	Returns UniDisk 3.5 status

Eject

The Eject extension to the Control call ejects a disk from the disk drive.

Control code

The control code for the UniDisk 3.5 Eject extension to the Control call is $04.

Control list

The control list for this call has a length of $0000.

Execute

The Execute extension to the Control call causes the intelligent controller in the UniDisk 3.5 disk drive to execute a 65C02 subroutine. The register setup is passed from the control list of this call to the routine to be executed. The address in the UniDisk 3.5 address space to which the routine is loaded is set with the SetAddress extension to the Control call. The 65C02 routine to be executed is downloaded with the DownLoad extension to the Control call.

Control code

The control code for the UniDisk 3.5 Execute extension to the Control call is $05.

Control list

You use the control list for this call to set the initial 65C02 register setup for the routine to be executed.

Count low byte	$06
Count high byte	$00
Accumulator value	$xx
X register value	$xx
Y register value	$xx
Processor Status value	$xx
Low program counter	$xx
High program counter	$xx

To reset the UniDisk 3.5 disk drive, use the Execute extension to the Control call with the program counter set to $E7A9. Doing so executes the UniDisk 3.5 disk drive's internal reset routine.

SetAddress

The SetAddress extension to the Control call sets the address in the UniDisk 3.5 controller memory space into which the DownLoad call loads a 65C02 routine. This address must point to free space in the UniDisk 3.5 memory map. The UniDisk 3.5 disk drive has free memory space from $0500 to $05FF, and has free zero page space from $00C0 to $00FF.

Control code

The control code for the UniDisk 3.5 SetAddress extension to the Control call is $06.

Control list

You use the control list for this call to set the starting address for the routine to be executed as follows:

Count low byte	$02
Count high byte	$00
Low byte address	$xx
High byte address	$xx

Download

The Download extension to the Control call downloads an executable 65C02 routine into the memory of the UniDisk 3.5 controller. The starting address to which the routine is loaded is set by the SetAddress extension to the Control call. The routine is executed by the Execute extension to the Control call.

Control code

The control code for the UniDisk 3.5 Download extension to the Control call is $07.

Control list

You use the control list for this call to list the 65C02 routine to be executed by the UniDisk 3.5 controller. The count field must be set to the length of the 65C02 routine to be downloaded. The control list is as follows:

Count low byte	$xx
Count high byte	$xx
Executable 65C02 routine	...

UniDiskStat

The UniDiskStat extension to the Status call allows an application to get more information about an error that occurs during a UniDisk 3.5 read or write operation. It also allows an application to read the 65C02 register state after the UniDisk 3.5 controller executes a 65C02 routine.

For more information about memory-mapped I/O addresses internal to the UniDisk 3.5 controller, see the SmartPort chapter in the *Apple IIGS Firmware Reference*.

Status code

The status code for the UniDisk 3.5 UniDiskStat extension to the Status call is $05.

Status list

The status list returned by this call is as follows:

Byte	$00
Soft error	$xx
Retries	$xx
A register after execute	$xx
X register after execute	$xx
P register after execute	$xx
Byte	$FF

An example of a SmartPort call in a program

The following text is an example of a SmartPort call in a program.

```
0000:          *
0000:          * This example shows how to find and use a SP interface.
0000:          * A search is made for a SP , and when one is found, a
0000:          * vector is set up that points to the SP entry. Then a
0000:          * Device Information Block (DIB) Status call is made, and
0000:          * if successful, the name string embedded in the DIB is
0000:          * output to the screen. Only the first device in the chain
0000:          * is accessed.
0000:          *
0000:          *
```

```
0000:                * Be sure to save any zero page locations that are in use
0000:                * before you execute this routine, and remember to restore
0000:                * them when you are done.
0000:                *
0000:                *
0000:                           MSB   ON
0000:                *
0000:                *
0000:          0006  ZPTempL    equ   $0006    ;Temporary zero
0000:                *                           page storage
0000:          0007  ZPTempH    equ   $0007
0000:                *
0000:          FDED  COut       equ   $FDED    ;Console output
0000:          FD8E  CROut      equ   $FD8E    ;Carriage return
0000:                *
0000:          0000  StatusCmd equ    0
0000:                *
0000:                *
0300:          0300             org   $300
0300:                *
0300:                * Find a SmartPort in one of the
0300:                * slots.
0300:                *
0300:20 43 03                   jsr   FindSP
0303:B0 1C     0321             bcs   Error
0305:                *
0305:                * Now make the DIB call to the first guy
0305:                *
0305:20 67 03                   jsr   Dispatch
0308:00                         dfb   StatusCmd
0309:6A 03                      dw    DParms
030B:B0 14     0321             bcs   Error
030D:                *
030D:                * Got the DIB; now print the name string
030D:                *
030D:A2 00                      ldx   #0
030F:          030F  morechars equ    *
030F:BD 74 03                   lda   DIBName,x
0312:09 80                      ora   #$80     ;COut wants high
0314                 *                           Bit set
0314:                *
0314:20 ED FD                   jsr   COut
```

```
0317:E8                      inx
0318:EC 73 03                cpx   DIBNameLen
031B:90 F2   030F            blt   morechars
031D:                *
031D:20 8E FD                jsr   CROut     ;Finish it off
0320:                *                         with a return
0320:                *
0320:60                      rts
0321:                *
0321:                *
0321:        0321      Error   equ   *
0321:                *
0321:                * There's either no SP around, or there
0321:                * was no Unit #1... give message
0321:                *
0321:A2 00                   ldx   #0
0323:        0323      errl    equ   *
0323:BD 2F 03                lda   Message,x
0326:F0 06   032E            beq   errout
0328:20 ED FD                jsr   COut
032B:E8                      inx
032C:D0 F5   0323            bne   errl
032E:                *
032E:        032E      errout  equ   *
032E:60                      rts
032F:                *
032F:CE CF A0 D0       Message asc   'NO SP OR NO DEVICE'
0341:8D 00                   dfb   $8D,0
0343:                *
0343:                *
0343:        0343      FindSP  equ   *
0343:                *
0343:                * Search slot 7 to slot 1 looking for
0343:                * signature bytes
0343:                *
0343:A2 07                   ldx   #7        ;Do for seven
0345:                *                         slots
0345:A9 C7                   lda   #$C7
0347:85 07                   sta   ZPTempH
0349:A9 00                   lda   #$00
034B:85 06                   sta   ZPTempL
034D:                *
```

```
034D:        034D  newslot   equ    *
034D:A0 07                   ldy    #7
034F:              *
034F:        034F  again     equ    *
034F:B1 06                   lda    (ZPTempL),y
0351:D9 70 03                cmp    sigtab,y      ;One of four
0354:              *                            byte signature
0354:F0 07   035D            beq    maybe         ;Found one
0356:              *                            signature byte
0356:C6 07                   dec    ZPTempH
0358:CA                      dex
0359:D0 F2   034D            bne    newslot
035B:              *
035B:              * If we get here, it's because we couldn't
035B:              * find a SmartPort.
035B:              * Exit with the carry set.
035B:              *
035B:38                      sec
035C:60                      rts
035D:              *
035D:              * If we get here, it means that one or
035D:              * more of the signature bytes
035D:              * for this card are what we're looking
035D:              * for. Decrement the byte
035D:              * counter and branch back to verify any
035D:              * remaining bytes.
035D:              *
035D:        035D  maybe     equ    *
035D:88                      dey
035E:88                      dey                  ;If N=1 then
035F:              *                         all sig bytes okay
035F:10 EE   034F            bpl    again
0361:              *
0361:              * Found a SmartPort interface.
0361:              * Set up the call address.
0361:              * We already have the high byte ($CN);
0361:              * we just need the low byte.
0361:              *
0361:        0361  foundSP   equ    *
0361:A9 FF                   lda    #$FF
0363:85 06                   sta    ZPTempL
0365:A0 00                   ldy    #0            ;For
```

```
0367:               *                         indirect load
0367:B1 06                    lda   (ZPTempL),y   ;Get the
0369:               *                              byte
0369:               *
0369:               * Now the Acc has the low order ProDOS
0369:               * entry point. The SP entry is
0369:               * three locations past this...
0369:               *
0369:18                       clc
036A:69 03                    adc   #3
036C:85 06                    sta   ZPTempL
036E:               *
036E:               * Now ZPTempL has the PC entry point.
036E:               * Return with carry clear.
036E:               *
036E:18                       clc
036F:60                       rts
0370:               *
0370:               *
0370:               * These are the SP signature bytes in
0370:               * their relative order.
0370:               * The $FF bytes are filler bytes and
0370:               * are not compared.
0370:               *
0370:FF 20 FF 00    sigtab    dfb   $FF,$20,$FF,$00
0374:FF 03 FF 00              dfb   $FF,$03,$FF,$00
0378:               *
0378:               *
0378:        0378   Dispatch equ    *
0378:6C 06 00                 jmp   (ZPTempL)     ;Simulate
037B:               *                    an indirect JSR to SP
037B:               *
037B:               *
037B:        037B   DParms    equ   *
037B:03             DPParmCt dfb   3              ;Status
037C:               *           calls have three parameters
037C:01             DPUnit    dfb   1
037D:80 03          DPBuffer dw    DIB
037F:03             DPStatCode dfb 3
0380:               *
0380:               *
0380:        0380   DIB       equ   *
```

```
0380:00              179 DIBStatByte1 dfb 0
0381:00 00 00        180 DIBDevSize dfb 0,0,0
0384:00              181 DIBNameLen dfb 0
0385:         0010   182 DIBName   ds    16,0
0395:00              183 DIBType   dfb   0
0396:00              184 DIBSubType dfb 0
0397:00 00           185 DIBVersion dw   0
0399:                186 *
0399:                187 *
```

Summary of commands and parameters

Table 8-7 summarizes the command numbers and parameter lists for SmartPort calls.

■ **Table 8-7** Summary of standard commands and parameter lists

Command	Status	ReadBlock	WriteBlock	Format	Control	Init	Open	Close	Read	Write
CMDNUM	$00	$01	$02	$03	$04	$05	$06	$07	$08	$09
CMDLIST										
byte										
0	$03	$03	$03	$01	$03	$01	$01	$01	$04	$04
1	Unit number	Unit number	Unit number	Unit number	Unit number	Unit number	Unit number	Unit number	Unit number	Unit number
2	Status list pointer	Data buffer pointer	Data buffer pointer		Control list pointer				Data buffer pointer	Data buffer pointer
3	Status list pointer	Data buffer pointer	Data buffer pointer		Control list pointer				Data buffer pointer	Data buffer pointer
4	Status code	Block number	Block number		Control code				Byte count	Byte count

■ **Table 8-7** Summary of standard commands and parameter lists (continued)

Command	Status	ReadBlock	WriteBlock	Format	Control	Init	Open	Close	Read	Write
5		Block number	Block number						Byte count	Byte count
6		Block number	Block number						*	*
7									*	*
8									*	*

* This parameter is device specific.

◆ *Note:* The Read byte count and the Control call list contents must not be larger than 767 bytes.

Upon return from the Read call, the byte count bytes contain the number of bytes actually read from the device.

Summary of SmartPort error codes

Following is a summary of error codes, including a brief description of the possible causes for each. If there is no error, the C flag (in the Status register of the microprocessor) is cleared (0), and the accumulator (the A register) contains 0's. If the call was unsuccessful, the C flag is set (1), and the A register contains the error code.

$00		No error.
$01	BadCmd	A nonexistent command was issued. Check the command number.
$04	BadPCnt	Bad call parameter count. The call parameter list was not properly constructed. Make sure the parameter list has the correct number of parameters.

$06	BusErr	A communications error between the device controller and the Apple IIc family member. Make sure that RAM is both read-enabled and write-enabled. Check the hardware (cables and connectors) between the device and the Apple IIc family member. Check for noise sources; make sure the cable is properly shielded.
$11	BadUnit	Unit number $00 was used in a call other than Status, Control, or Init.
$21	BadCtl	The control code or status code is not supported by the device.
$22	BadCtlParm	The Control parameter list contains invalid information. Make sure each value is within the range allowed for that parameter.
$27	IOError	The device encountered an I/O error when trying to read or write to the recording medium. Make sure that the medium in the device is formatted and not defective. Make sure the device is operating correctly.
$28	NoDrive	The device is not connected. This can occur if there is no device with the unit number specified.
$2B	NoWrite	The medium in the device is write-protected.
$2D	BadBlock	The block number is outside the range allowed for the medium in the device. This range depends on the type of device and the type of medium in the device (single-sided versus double-sided disk, for example).
$2F	OffLine	Device off line or no disk in drive. Check the cables and connections; make sure the medium is present in the drive, and that the drive is functioning correctly.
$30–$3F	DevSpec	Errors that differ from device to device. See the technical manual for the device in question for details.
$40–$4F	Reserved	Reserved for future expansion.
$50–$7F	NonFatal	A device-specific soft error. The operation completed successfully, but some exception condition was detected. For details, see the technical manual for the device in question.

Chapter 9 **Mouse and Game Input**

This chapter describes the mouse port and hand control (game) input capabilities of the Apple IIc. The mouse and hand controls use the same 9-pin connector on the back panel; the firmware uses the port as directed by keyboard or program commands. ■

△ Important — Mouse firmware entry points vary between versions of the Apple IIc. Thus, *all port-dependent mouse addresses are given as $Cnxx where n is the port* number. Port numbers for the various versions of the Apple IIc are

- Original and UniDisk 3.5 versions: n=4
- Memory expansion version and Apple IIc Plus: n=7 △

Mouse input

Table 9-1 summarizes the mouse port's characteristics and guides you to other information in this part of the chapter.

▲ **Warning** — If you want your programs that use the mouse on the Apple IIe and other Apple II–series *computers to work with the Apple IIc family*, always use the I/O firmware entry points listed in Tables 9-3 and 9-4, rather than directly addressing the mouse hardware and RAM locations. ▲

The mouse back panel connector is described in Chapter 11.

■ **Table 9-1** Mouse input port characteristics

Port number	original/UniDisk 3.5 port 4 memory expansion/Apple IIc Plus port 7

BASIC commands	Turn on mouse*:
	`PRINT CHR$(4)"PR#n":PRINT CHR$(1)`
	Turn off mouse interrupts*:
	`PRINT "PR#n":PRINT CHR$(0)`
	Turn on graphics character set: See "MouseText" in Chapter 6.

■ **Table 9-1** Mouse input port characteristics (continued)

Initial characteristics	After a reset, all mouse interrupts are off and the rising edge of X0 and Y0 are selected for interrupts. See "Mouse Input" in Chapter 11.
Hardware page locations	See Table 9-2.
Monitor firmware routines	None.
I/O firmware entry points	See Tables 9-3 and 9-4.
Use of screen holes	See Table 9-5.

* *n* is the port number of the mouse (4 or 7).

Mouse connector signals

The mouse uses the same DB-9 connector as the hand controls. However, the interpretation of the signals arriving on the pins differs depending on the commands and signals received. The connector pinout for the mouse port is given in Chapter 11.

Mouse operating modes

Your program should call the ServeMouse routine to determine the source of an interrupt as soon as it receives one in all the interrupt modes except transparent mode. Interrupts are discussed in Chapter 3.

△ Apple IIe — There are some important differences between how the mouse works with the Apple IIc family computers and with an Apple IIe with a mouse card. These differences are discussed in the section "Mouse Input" in Appendix D. △

Transparent mode

In transparent mode, your program must read screen holes to check for mouse movement. An interrupt routine in the Apple IIc–family firmware updates mouse position counters each time the mouse is moved, then returns control to the main program task. The findings of the interrupt routine are placed in the screen holes where your program can read them. Table 9-5 lists the screen holes with the information that your program should look for.

This is the only mouse mode available to BASIC programs.

Movement interrupt mode

Whenever the mouse moves, it sends a signal to the IOU, as described in the section "Mouse Input," in Chapter 11. The IOU sends an interrupt to the processor the next time a vertical blanking (VBL) signal occurs, and the processor executes the system interrupt handler routine. (In the Apple IIc family, the video circuitry generates a vertical blanking signal 60 times each second.) When movement interrupt mode is active, the system interrupt handler checks the mouse status bits (described in Table 9-2) for mouse movement. If the mouse has moved, the firmware updates the mouse-movement screen holes and then passes control to the user interrupt handler. (The location of the user interrupt handler is stored in the user interrupt vector at $03FE–$03FF.) The user interrupt handler can then update the cursor to its next screen position.

Your interrupt handler must first call ServeMouse (described in Table 9-3) to see if the mouse caused the interrupt. It should then call ReadMouse to get the mouse status and its current X-Y position. Your interrupt handler can also change the mouse mode if you so desire. For example, you might want to change to a vertical-blanking-active mode so that your interrupt handler gets control 60 times every second.

If your program initializes the mouse and sets bit 7 of the mouse port mode byte at $07FF ($07FC in the UniDisk 3.5 version) to 1, mouse movement interrupts are passed directly to the user interrupt handler. In this case, the firmware does not check the source of the interrupt, does not read the mouse status bits, and does not update the screen holes. VBL interrupts are still handled by the system interrupt handler. You should use this feature only if the mouse firmware can't keep up with your needs. You can use the SetMouse routine described in Table 9-3 to set the mode byte. This feature is not available in the original Apple IIc.

Button interrupt mode

The Apple IIc–family mouse-button hardware does not generate interrupts. However, in button interrupt mode, the system interrupt handler polls the mouse button every time a VBL signal occurs and passes control to the user interrupt handler if the mouse button has been pressed. Your interrupt handler must first call ServeMouse (Table 9-3) to see if the mouse button caused the interrupt. It should then call ReadMouse to get the mouse button status.

Movement/button interrupt mode

The movement/button interrupt mode is a combination of the two modes just described. It provides the best response possible without constant polling of the mouse position and button. By using this mode, your program can effectively process a main task concurrently with cursor and menu updating.

Vertical-blanking-active modes

When enabled by the EnVBl soft switch (Table 9-2), an IRQ interrupt is sent to the processor each time the VBL signal is asserted. When you select a vertical-blanking-active mouse mode, the firmware enables vertical blanking interrupts and the system interrupt handler passes control to the user interrupt handler each time a vertical blanking interrupt occurs. Each time a VBL interrupt occurs, the vertical blanking interrupt flag is set; you can read and reset this flag by using the RstVBl soft switch shown in Table 9-2.

In all other respects (other than whether VBL interrupts are enabled and passed on to the user interrupt handler), the VBL-active mouse modes are identical to the VBL-inactive mouse modes.

Mouse soft switches and status bits

The soft switches assigned to the mouse interface are shown in Table 9-2. On power-up or reset, the hardware selects the rising edge of X0 and Y0 (mouse movement signals) and masks out all mouse interrupts.

This section tells what the mouse operating modes are for. Later sections of this chapter describe how to set the various mouse operating modes.

▲ **Warning** Table 9-2 is included here for your information only. You should use the built-in firmware to access the mouse. Writing your own mouse interrupt handler virtually assures that your program will not work with future versions of the Apple II. ▲

■ **Table 9-2** Mouse soft switches and status bits

Name	Action	Hex	Function
IOUDis	W	$C07E	On: Disable IOU access for addresses $C058 to $C05F; enable access to DHiRes switch*
IOUDis	W	$C07F	Off: Enable IOU access for addresses $C058 to $C05F; disable access to DHiRes switch*
RdIOUDis	R7	$C07E	Read IOUDis switch (1 = off)†
DisXY	R/W	$C058	Disable (mask) X0 and Y0 movement interrupts‡
EnbXY	R/W	$C059	Enable (allow) X0 and Y0 movement interrupts‡
RdXYMsk	R7	$C040	Read status of X0/Y0 interrupt mask (1 = mask on)
RstXY	R	$C048	Reset X0/Y0 interrupt flags
X0Edge	R/W	$C05C	Select rising edge of X0 for interrupt‡
X0Edge	R/W	$C05D	Select falling edge of X0 for interrupt‡
RdX0Edge	R7	$C042	Read status of X0 edge selector (1 = falling)
RstXInt	R	$C015	Reset mouse X0 interrupt flag
Y0Edge	R/W	$C05E	Select rising edge of Y0 for interrupt‡
Y0Edge	R/W	$C05F	Select falling edge of Y0 for interrupt‡
RdY0Edge	R7	$C043	Read status of Y0 edge selector (1 = falling)
RstYInt	R	$C017	Reset mouse Y0 interrupt flag
DisVBl	R/W	$C05A	Disable (mask) VBL interrupts‡
EnVBl	R/W	$C05B	Enable (allow) VBL interrupts‡
RdVBlMsk	R7	$C041	Read status of VBL interrupt mask (1 = mask on)
RstVBl	R	$C019	Read and then reset VBL interrupt flag
PTrig	R/W	$C070	Reset VBlInt flag; trigger paddle timer
RdBtn0	R7	$C061	Read first mouse button status (1 = pressed)§
Rd63	R7	$C063	Read second mouse button status (0 = pressed)¶

■ **Table 9-2** Mouse soft switches and status bits (continued)

Name	Action	Hex	Function
MouX1	R7	$C066	Read status of X1 (mouse X direction) (1 = high)
MouY1	R7	$C067	Read status of Y1 (mouse Y direction) (1 = high)

* When IOUDis is on, $C058–$C05F do not affect mouse, and $C05E and $C05F become DHiRes (Table 6-9).
† Read or write to $C07n also resets the VBL interrupt and triggers paddle timers.
‡ These work only if IOUDis is off.
§ This location is also the Command key (Table 5-1).
¶ This is also the location of the Shift-key mod (Appendix D).

Software can select which edge of X0 and Y0 information will cause the XInt or YInt.

When an interrupt has occurred, you can read the direction of the mouse's X1 movement by reading address $C066 bit 7, and Y1 movement by reading address $C067 bit 7.

A program can read the status of the soft switches by reading one of the locations $C040–$C043 and then testing data bit 7. The soft switches are described in Table 9-2.

The section on mouse input in Chapter 11 explains what X0, Y0, X1, Y1 are and what they mean with respect to mouse movement.

I/O firmware support for mouse input

The Apple IIc family supports the mouse with firmware starting at address $Cn00. This firmware is necessary because the mouse requires fast, transparent interrupt processing to work effectively.

In assembly language you can use direct firmware support for sophisticated mouse applications. To enable the mouse, first load a mode byte into the accumulator (and $Cn in X, $n0 in Y) and then do a JSR to the firmware routine called *SetMouse* (described in Table 9-3). Valid mode bytes are the following:

$00	Turns mouse off
$01	Sets transparent mode
$03	Sets movement interrupt mod
$05	Sets button interrupt mode
$07	Sets movement or button interrupt mode
$08	Turns mouse off, VBlInt active
$09	Sets transparent mode, VBlInt active
$0B	Sets movement interrupt mode, VBlInt active
$0D	Sets button interrupt mode, VBlInt active
$0F	Sets movement or button interrupt mode, VBlInt active

The firmware then initializes the mouse. To read the current position and status of the mouse, first load $Cn into the X register, load $n0 into the Y register, save processor status, disable interrupts, and then perform a JSR to the firmware routine called *ReadMouse* (described in Table 9-3), which stores the information in the port *n* screen holes (listed in Table 9-5).

Table 9-3 lists the locations at which you can read the offsets of the mouse port firmware routines. Each offset in Table 9-3 contains the low byte of the entry point of the routine described. The high byte of the routine is $Cn. The calling setup for all routines (except ServeMouse) is the same: the X register must contain $Cn, and the Y register must contain $n0. When the routine has finished, the A, X, and Y register contents are undefined.

◆ *Note:* The mouse port firmware routines in the Apple IIc and Apple IIc Plus computers do not require that specific values be set in the X and Y registers; however, you should always follow these directions to maintain compatibility with other Apple II family computers and with future versions of the Apple IIc.

■ **Table 9-3** Offsets for pointers to Mouse firmware routines

Offset from $Cn00*	Routine	Description
$12	SetMouse	Sets the mouse mode to the value in the accumulator. Input: A register contains mode (see $07Fx, Table 9-5) Output: Carry bit = 0 means mode was legal; carry bit = 1 means mode was not legal.
$13	ServeMouse	Services mouse interrupt if needed. Input: X, Y, A registers—doesn't matter. Output: Carry bit = 0 means mouse caused the interrupt; carry bit = 1 means something else caused it. This routine updates $077x to show which event caused the interrupt (see Table 9-5).
$14	ReadMouse	Updates screen holes to show current mouse X-Y position and button status; clears VBlInt, button and movement interrupt bits in the status byte. Doesn't reenable interrupts until after retrieving position values. Output: Carry bit = 0.
$15	ClearMouse	Sets the mouse position to 0, though not necessarily within clamping boundaries; leaves button and interrupt bits in status byte unchanged. Output: Carry bit = 0.
$16	PosMouse	Sets the mouse coordinates to new values. Input: X and Y screen holes contain new X and Y positions. Output: Carry bit = 0.
$17	ClampMouse	Sets new clamping boundaries (see Table 9-5). Does not affect mouse position or update mouse position screen holes; use ReadMouse to do that. Input: A register = 0 means set new X boundaries; A register = 1 means set new Y boundaries. Output: Carry bit = 0.
$18	HomeMouse	Sets the internal mouse position to the upper-left corner of the clamping window. Does not update mouse position screen holes; use ReadMouse to do that.
$19	InitMouse	Sets startup internal values; does not update mouse-position screen holes. Output: Carry bit = 0.

* The value located at this offset is the low byte of the address of the routine. The high byte is C n.

Note: n = 4 for the original Apple IIc and the UniDisk 3.5 Apple IIc; n = 7 for the memory expansion Apple IIc and the Apple IIc Plus.

Here is a sample sequence of events and calls:

1. Four screen holes contain the mouse's X and Y coordinates, and one contains the status of the last mouse movement (see Table 9-5).
2. Call InitMouse.
3. Inhibit interrupts, set up the boundaries you want, then call ClampMouse.
4. Use PosMouse, HomeMouse, or ClearMouse to position the mouse where you want it.
5. Put the mouse mode (see address $07Fx in Table 9-5) that you want to use in the accumulator, then call SetMouse.
6. If you have set one of the interrupt modes, then when an interrupt arrives, call ServeMouse to determine the source of the interrupt.
7. Disable interrupts and call ReadMouse. Retrieve the position values, then reenable interrupts.

Pascal support

Table 9-4 lists the locations and values of the I/O firmware protocol that Pascal 1.1 and later versions use. However, Pascal must use a special routine (called an *attach driver*) to support the mouse.

■ **Table 9-4** Mouse port I/O firmware protocol for Pascal

Address	Value	Description
$Cn05	$38	Pascal ID byte
$Cn07	$18	Pascal ID byte
$Cn0B	$01	Generic signature byte of firmware cards
$Cn0C	$20	2 = X-Y pointing device; 0 = identification code
$Cn0D		Initialization routine (not implemented; returns error code)
$Cn0E		Standard read routine (not implemented; returns error code)
$Cn0F		Standard write routine (not implemented; returns error code)
$Cn10		Standard status routine (not implemented; returns error code)
$Cn11	$00	Optional routines follow
$CnFB	$D6	A mouse identification byte

BASIC and assembly-language support

In BASIC you must turn the mouse on by printing PR#*n* and then CHR$(1) before you can get input from the mouse. This sets transparent mode. After that, reenable video output with PR#3 and take subsequent input from the mouse by issuing IN#*n*. The first input statement after that (INPUT X,Y,S) initializes and enables the mouse and returns a three-element string

```
+xxxx,+yyyy,+st
```

representing the x-coordinate, y-coordinate, and status digits.

The coordinates will be integers between 0 and +1023. These are called the *clamping boundaries* of the mouse.

The sign preceding the status digits is normally positive; it becomes negative when you press a key on the keyboard.

The first digit, s, of the status is 0. The second digit, t, of the status is 1 if the mouse button is still pressed, 2 if it was just pressed, 3 if it was just released, and 4 if it is still released.

To disable the mouse, use these statements:

```
PRINT CHR$(4)"PR#n"
PRINT CHR$(0)
PRINT CHR(4)"PR#3"
```

Screen holes

Table 9-5 lists the screen holes that the mouse firmware uses. Note that the mouse firmware reserves port 5 screen holes for its own use. Also, the auxiliary page counterparts of the port 4 addresses are reserved for startup values.

■ **Table 9-5** Mouse port screen hole locations

Location	Description
Scratch area	
$0478	Low byte of clamping minimum
$04F8	Low byte of clamping maximum
$0578	High byte of clamping mimimum
$05F8	High byte of clamping maximum
Port 4 screen holes*	
$047x	Low byte of X coordinate
$04Fx	Low byte of Y coordinate
$057x	High byte of X coordinate
$05Fx	High byte of Y coordinate
$067x	Reserved
$06Fx	Reserved
$077x	Status byte
Bit	**1 Equals**
7	Button down
6	Button was down on last read and is still down
5	Movement since last read
4	Reserved
3	Interrupt from VBlInt
2	Interrupt from button
1	Interrupt from movement
0	Reserved
$07Fx	Mode byte (current mode; mask out bits 4–7 when testing)
Bit	**1 Equals**
7	Use user interrupt handler (not available in original Apple IIc)
6-4	Reserved
3	VBlInt active
2	Interrupt on mouse button
1	Interrupt on mouse movement
0	Mouse active
Port 5 screen holes	
Reserved	

* x is C for the original and UniDisk 3.5 versions of the Apple IIc, and F for the memory expansion version and the Apple IIc Plus.

Using the mouse as a hand control

You can use the mouse as if it were a set of hand controls or an X-Y pointing device. If you turn the mouse on, the Monitor hand control (game paddle) routines take input from the mouse. This is possible because the mouse and the hand controls use the same back panel connector.

You can run a BASIC program that uses the Pdl function to read from the mouse by doing the following, either from the keyboard or from a program:

1. Start up the system with the BASIC program that uses paddles.
2. Type `PR#`*n* and press Return to turn on the mouse. The number *n* is 4 for the original and UniDisk 3.5 versions of the Apple IIc, and 7 for the memory expansion version and the Apple IIc Plus.
3. Press Control-A Return to initialize the mouse.
4. Type `PR#0` and press Return to restore output to the screen.
5. Run the program.

Play the game using the mouse instead of the paddles.

△ Important   Many copy-protected games do not work with a mouse. In addition, many games don't use built-in firmware for the paddles. △

Game input

The Apple IIc family supports game paddles, joysticks, and other hand controls connected to the DB-9 connector on its back panel. Table 9-6 is a summary of game input characteristics.

■ **Table 9-6** Game input characteristics

Port number	None.	
Commands	None.	
Initial characteristics	Game inputs cannot be disabled.	
Hardware page locations		
$C061	Paddle 0 button, mouse button, and Command (1 = pressed).	
$C062	Paddle 1 button and Option key (1 = pressed).	
$C063	Mouse button (0 = pressed).	
$C064	Analog input (paddle) 0.	
$C065	Analog input (paddle) 1.	
$C070	Trigger paddle timer.	
Monitor firmware routine		
Location	**Name**	**Description**
$FB1E	PRead	Reads a paddle position.
I/O firmware entry points	None.	
Use of screen holes	None.	

The hand control connector signals

Several inputs are available to programs or devices from the DB-9 connector on the back of each Apple IIc family member: two 1-bit inputs, or *switches*, and two analog inputs. The pinouts for the hand control connector are given in the section "Hand Control Input," in Chapter 11.

When you connect a pair of hand controls to the 9-pin connector, the rotary controls use two analog inputs, and the pushbuttons use two 1-bit inputs. However, you can also use these inputs for many other jobs. For example, two analog inputs can be used with a two-axis joystick.

Switch inputs (Sw0 and Sw1)

The two 1-bit inputs can be connected to the output of another electronic device that meets the electrical requirements described in Chapter 11, or to a pushbutton. When you read a byte from one of these locations, only the high-order bit—bit 7—is valid information; the rest of the byte is undefined. From machine language, you can do a Branch Plus or Branch Minus on the state of bit 7. From BASIC, you can read the switch with a PEEK instruction and compare the value with 128. If the value is 128 or greater, the switch is on.

The addresses for these switches are $C061, $C062, and $C063 (decimal locations 49249 through 49251), as shown in Table 9-6. The lines from switch 0 and switch 1 (the buttons on the hand controls) are hard wired to the Command and Option keys (Open Apple and Closed Apple) on the keyboard. The switch 0 line is also connected to the mouse button. Location $C063 is a second address for the mouse button; this address is provided so that a program can distinguish a mouse button from the Command key. When the mouse button is pressed, $C063 (bit 7) goes from 1 to 0, and $C061 (bit 7) goes from 0 to 1.

Analog inputs (Pdl0 and Pdl1)

The two analog inputs are designed for use with 150-KΩ variable resistors or potentiometers. The variable resistance is connected between the +5V supply and each input, so that it makes up part of a timing circuit. The circuit changes state when its time constant has elapsed, and the time constant varies as the resistance varies. Your program can measure this time by counting in a loop until the circuit changes state, or times out.

A program must first reset the timing circuits before it can read the analog inputs. Accessing memory location $C070 does this. As soon as you reset the timing circuits, the high bits of the bytes at locations $C064 through $C067 are set to 1. If you use a PEEK instruction in BASIC to determine the values of locations 49252 through 49255, the values will be 128 or greater. Within about 3 milliseconds, these bits will change back to 0—byte values less than 128—and remain there until you reset the timing circuits again. The exact amount of time each of the bits remains high is directly proportional to the resistance connected to the corresponding input. If these inputs are open—no resistances are connected—the corresponding bits may remain high indefinitely.

Monitor support for game input

To read the analog inputs from machine language, you can use a program loop that resets the timers and then increments a counter until the bit at the appropriate memory location changes to 0, or you can use the built-in routine PRead. BASIC and other high-level languages also include convenient means of reading the analog inputs—refer to your language manuals. You can read and reread the same paddle at arbitrarily short intervals. However, you must wait at least 3 milliseconds between reading one paddle and reading a different paddle.

The Monitor routine PRead (at address $FB1E) places in the Y register a number between $00 and $FF that represents the position of a hand control. You pass the number of the hand control in the X register.

▲ **Warning** If the hand control number you furnish in the X register does not equal 0 or 1, strange things may happen.

The paddle and vertical blanking both use $C070. Disable interrupts before calling PRead if you are reading the paddles and using VBL interrupts. ▲

Chapter 10 **Using the Monitor**

The System Monitor is a set of subroutines in the Apple IIc–family firmware that provides a standard interface to the built-in I/O devices. Many of the I/O subroutines described in Chapters 4 through 9 are part of the System Monitor. ■

DOS, ProDOS, and the BASIC interpreters use Monitor subroutines by direct calls to their starting locations. You can call many of the standard subroutines from your programs in the same fashion. All of the standard firmware routines that Apple Computer, Inc., guarantees to maintain are described in Appendix F.

You can perform most of the Monitor functions directly from the keyboard. This chapter tells you how to use the Monitor

- to look at one or more memory locations
- to change the contents of any location
- to write small programs in machine language to be executed directly by the Apple IIc family computer
- to move and compare blocks of memory
- to invoke other programs from the Monitor

Invoking the Monitor

The System Monitor starts at memory location $FF69 (–151). To invoke the Monitor, make a CALL –151 statement to this location from the keyboard or from a BASIC program. When the Monitor is running, its prompt character, an asterisk (*), appears on the left side of the display screen, followed by a cursor.

To use the Monitor, type commands at the keyboard. When you have finished using the Monitor, you return to the BASIC language you were previously using by pressing Control-Reset, by pressing Control-C and then Return, or by typing `3D0G`, which executes the resident program—usually Applesoft—whose address is stored in a jump instruction at location $03D0.

△ Important — If ProDOS (or DOS) is connected via the standard I/O links (described in Chapter 4), you can issue commands to it from the Monitor. Under this arrangement, errors will return control to BASIC rather than to the Monitor. △

If you want to have Control-Reset return you to the Monitor, load the values \$69, \$FF, and \$5A (decimal 105, 255, and 90) into the three locations starting at address \$03F2 (decimal 1010, the reset-vector address, and the power-up byte).

Syntax of Monitor commands

To give a command to the Monitor, type a line on the keyboard, then press Return. The Monitor accepts the line using the standard I/O subroutine GetLn described in Chapter 4. A Monitor command can be up to 255 characters in length, ending with a carriage return. It can include three kinds of information: addresses, data values, and command characters. You type addresses and data values in hexadecimal notation.

When the command you type calls for an address, the Monitor accepts any group of hexadecimal digits. If there are fewer than four digits in the group, it adds leading 0's; if there are more than four hexadecimal digits, the Monitor uses only the last four digits. It follows a similar procedure when the command syntax calls for two-digit data values.

Each command you type consists of one command character, usually the first letter of the command name. The Monitor recognizes 22 different command characters. Some of them are punctuation marks, some are letters (uppercase or lowercase), and some are control characters.

◆ *Note:* Although the Monitor recognizes and interprets control characters in an input line, they do not appear on the screen.

This chapter contains examples of Monitor command use. Some of the data values displayed by your Apple IIc–family computer may differ from the values printed in these examples, because they are variables stored in RAM.

Monitor memory commands

When you use the Monitor to examine and change the contents of memory, it keeps track of the address of the last location whose value you inquired about and the address of the location that is next to have its value changed. These are called the *last opened location* and the *next changeable location*.

▲ **Warning** Because locations $C000 through $C0FF contain special hardware circuits, issuing any command that reads or writes on this page can have unpredictable, and perhaps disastrous, results. ▲

Examining memory contents

When you type the address of a memory location and press Return, the Monitor responds with the address you typed, a dash, a space, and the value stored at that location, like this:

```
*E000
E000- 4C
*33
0033- AA
*
```

Each time the Monitor displays the value stored at a location, it saves that address as the last opened location and as the next changeable location.

Memory dump

When you type a period (.) followed by an address, and then press Return, the Monitor displays a memory dump: the data values stored at all the memory locations from the one following the last opened location to the location whose address you typed following the period. The Monitor saves the last location displayed as both the last opened location and the next changeable location. In these examples, the amount of data displayed by the Monitor depends on how much larger the address after the period is than the last opened location.

```
*20
0020- 00
*.2B
0021- 28 00 18 0F 0C 00 00
0028- A8 06 D0 07
*300
0300- 99
*.315
0301- B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03
*.32A
0316- 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20
*
```

When the Monitor performs a memory dump, it starts at the address immediately following the last opened location and displays that address and the data value stored there. It then displays the values of successive locations up to and including the location whose address you typed, but only up to eight values on a line. When it reaches a location whose address is a multiple of 8—that is, one that ends with an 8 or a 0—it displays that address as the beginning of a new line, then continues displaying more values.

After the Monitor has displayed the value at the location whose address you specified in the command, it stops the memory dump and sets that location as both the last opened location and the next changeable location. If the address specified on the input line is less than the address of the last opened location, the Monitor displays only the address and value of the location following the last opened location.

You can combine the two commands, opening a location and dumping memory, by concatenating them: type the first address, a period, and the second address. This combination of two addresses separated by a period is called a *memory range*.

```
*300.32F
0300- 99 B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20 46 03 A5 3D 4D
*30.40
0030- AA 00 FF AA 05 C2 05 C2
0038- 1B FD D0 03 3C 00 40 00
0040- 30
*E015.E025
E015- 4C ED FD
E018- A9 20 C5 24 B0 0C A9 8D
E020- A0 07 20 ED FD A9
*
```

Pressing Return by itself makes the Monitor display one line of a memory dump; that is, a memory dump from the location following the last opened location to the next multiple-of-eight boundary. The Monitor saves the address of the last location displayed as both the last opened location and the next changeable location.

```
*5
0005- 00
*Return
00 00
*Return
0008- 00 00 00 00 00 00 00 00
*32
0032- FF
*Return
AA 00 C2 05 C2
*Return
0038- 1B FD D0 03 3C 00 3F 00
*
```

Changing memory contents

The section on memory dumping showed you how to *display* values stored in the Apple IIc family computer's memory; this section shows you how to *change* these values. You can change any location in RAM; you can change the characteristics and treatment of an output device by changing the contents of locations assigned to it; and you can change a soft switch setting by referencing its set and reset addresses.

▲ **Warning** Use these commands carefully. If you change the zero-page locations used by the interpreter or operating system, you may lose programs or data stored in memory. ▲

Changing one byte

The previous commands keep track of the next changeable location; these commands make use of it. In the next example, which illustrates the Store (:) command, you open location 0, then type a colon followed by a value.

```
*0
0000- 4C
*:5F
```

The contents of the next changeable location have just been changed to the value you typed, as you can see by examining that location:

```
*0
0000- 5F
*
```

You can also combine opening and changing into one operation by typing an address followed by a colon and a value. In the next example, you type the address again to verify the change.

```
*302:42
*302
0302- 42
*
```

When you change the contents of a location, the value that was contained in that location is replaced by the new value, which will remain until you or some program replaces it with another value.

- *ASCII input mode:* The Monitor has a tool to make entering values a little easier: ASCII input mode. ASCII input mode lets you enter ASCII characters as well as their hexadecimal ASCII equivalents. The Monitor uses the ASCII value of any character following an apostrophe; for example, 'A is the same as C1 and 'B is the same as C2 to the Monitor. Therefore, to enter the string "Good morning!" at $0300 in memory, type

  ```
  *300:'G 'o 'o 'd ' 'm 'o 'r 'n 'i 'n 'g '!
  ```

  Note that each character to be placed in memory is delimited by a leading and a trailing space. The only exception to this rule is that the last character in the line is followed by Return instead of a space.

Changing consecutive locations

You don't have to type a separate command with an address, a colon, and a value, and press Return for each location you want to change. You can change the values of up to 85 consecutive locations at a time—or even more, if you omit leading 0's from the values—by typing only the initial address and colon followed by all the values separated by spaces; end with Return. The Monitor stores the consecutive values in consecutive locations, starting at the location whose address you typed. After it has processed the string of values, it takes the location following the last changed location as the next changeable location. Thus, you can continue changing consecutive locations, without typing an address on the next input line, by typing another colon and more values.

In these examples, you first change some locations, then examine them to verify the changes.

```
*300:69 01 20 ED FD 4C 03
*300
0300- 69
*Return
01 20 ED FD 4C 00 03
*10:0 1 2 3
*:4 5 6 7
*10.17
0010- 00 01 02 03 04 05 06 07
*
```

Moving data in memory

You can copy a contiguous block of data from one area to another in RAM by using the Monitor's Move command. To move a range of memory, you must tell the Monitor both where the data is now situated in memory—the source locations—and where you want the copy to go—the destination locations.

The format of the complete Move command looks like this:

{*destination*} < {*start*} . {*end*} M

The destination is the address where you want the first of the moved data to go. The less-than symbol (<) separates the destination address from the starting and ending addresses of the block of data to be moved. The period between two addresses is the Monitor's standard notation for specifying address ranges. If the second address in the source range specification is less than the first, only one value (that of the first location in the range) will be moved.

When you type the actual command, replace the words in braces with hexadecimal addresses and omit the braces and spaces.

Here are some examples of memory moves. First examine the values stored in one range of memory, then store several values in another range of memory. The actual Move commands end with M.

```
*0.F
0000- 5F 00 05 07 00 00 00 00
0008- 00 00 00 00 00 00 00 00
*300:A9 8D 20 ED FD A9 45 20 DA FD 4C 00 03
*300.30C
0300- A9 8D 20 ED FD A9 45 20
0308- DA FD 4C 00 03
*0<300.30C M
*0.C
0000- A9 8D 20 ED FD A9 45 20
0008- DA FD 4C 00 03
*310<8.A M
*310.312
0310- DA FD 4C
*2<7.9 M
*0.C
0000- A9 8D 20 DA FD A9 45 20
0008- DA FD 4C 00 03
*
```

The Monitor moves a copy of the data stored in the source range of locations to the destination locations. The values in the source range are left undisturbed. The Monitor remembers the last location in the source range as the last opened location, and the first location in the source range as the next changeable location.

If the destination address of the Move command is inside the source range of addresses, then strange things happen: the locations between the beginning of the source range and the destination address are treated as a subrange and the values in this subrange are replicated throughout the source range. Try it.

Comparing data in memory

You can use the Verify command to compare two ranges of memory using the same format you use to move a range of memory from one place to another. In fact, the Verify command can be used immediately after a Move command to make sure that the move was successful.

The Verify command, like the Move command, needs a range and a destination. The syntax of the Verify command is

{*destination*} < {*start*} . {*end*} V

The Monitor compares the values in the source locations with the values in the locations beginning at the destination address. If any values don't match, the Monitor displays the address at which the discrepancy was found and the two values that differ. In the example, you store data values in the range of locations from 0 to $0D, copy them to locations starting at $0300 with the Move command, and then compare them using the Verify command. When you use the Verify command after you change the value at location 6 to $E4, the Monitor detects the change.

```
*0:D7 F2 E9 F4 F4 E5 EE A0 E2 F9 A0 C3 C4 C5
*300<0.D M
*300<0.D V
*6:E4
*300<0.D V
0006-E4 (EE)
*
```

If the Verify command finds a discrepancy, it displays the address of the location in the source range whose value differs from its counterpart in the destination range. If there is no discrepancy, Verify displays nothing. The Verify command leaves the values in both ranges unchanged. The last opened location is the last location in the source range, and the next changeable location is the first location in the source range, just as in the Move command. If the ending address of the range is less than the starting address, the values of only the first locations in the ranges are compared. Like the Move command, the Verify command does unusual things if the destination address is within the source range; see "Advanced Operations" later in this chapter.

Monitor register commands

Even though the actual contents of the 65C02's internal registers are changing as you use the Monitor, you can examine the values that the registers contained at the time the Monitor gained control, either because you called it or because the program you are debugging stopped at a break (BRK). You can also store new register values that will be used when you execute a program from the Monitor using the Go command, described in the section "Running a Program," later in this chapter.

Changing registers

When you call the Monitor, it stores the contents of the 65C02 registers in memory. The registers are stored in the order A, X, Y, P (Processor Status register), and S (stack pointer), starting at location $45. When you give the Monitor a Go command, the Monitor loads the registers from these five locations before it executes the first instruction in your program.

Examining registers

Pressing Control-E and then Return invokes the Monitor's Examine command, which displays the stored register values and sets the location containing the contents of the A register as the next changeable location. After using the Examine command, you can change the values in these locations by typing a colon and then typing the new values separated by spaces. In the following example, you display the registers, change the first two, and then display them again to verify the change.

```
*Control-E
M=00 A=0A X=FF Y=D8 P=B0 S=F8
*:B0 02
*Control-E
M=00 A=B0 X=02 Y=D8 P=B0 S=F8
*
```

In the Examine command's display, M shows the current memory state register contents. The memory state register is location $44, and its interpretation is given in the section "Handling Break Instructions" in Chapter 3.

Miscellaneous Monitor commands

Monitor commands discussed in this section let you do the following:

- change the video display format from normal to inverse and back
- assign input and output to various devices
- leave the Monitor and return to the currently loaded operating system (DOS 3.3 or ProDOS) or BASIC

Display inverse and normal

You can control the setting of the inverse-normal mask location used by the COut routine from the Monitor so that all the Monitor's output will be in inverse format. The Inverse command (I) sets the mask so that all subsequent inputs and outputs are displayed in inverse format. To switch the Monitor's output back to normal format, use the Normal command (N). The COut routine is described in the section "COut Subroutine," in Chapter 4.

```
*0.F
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
*I
```

```
*0.F
0000- 0A 0B 0C 0D 0E 0F D0
0008- C6 01 F0 08 CA D0 F6
*N
```

```
*0.F
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
*
```

Back to BASIC

If you are using one of the Apple disk operating systems (ProDOS or DOS), press Control-Reset or type `3D0G` to return to the language you were using, with your program and variables intact.

△ Important If you type `3D0G`, make sure that the third character you type is a zero, not a letter *O*. The letter *G* is the Monitor's Go command. △

If there is no operating system in RAM, use the BASIC command Control-B to leave the Monitor and enter the BASIC interpreter that was active when you entered the Monitor. (Normally this is Applesoft BASIC.) Any program or variables that you previously had in BASIC will be lost. If you want to reenter BASIC with your previous program and variables intact, use the Continue BASIC command (Control-C).

Redirecting input and output

The Control-P command diverts all output normally destined for the screen (port 0) to a device attached to one of the other ports, from 1 to 7. The format of the command is

{*port number*} Control-P

A Control-P command to port number 0 switches the stream of output characters back to the video display. However, use Esc Control-Q if the enhanced video firmware is active (solid-block cursor).

Control-K controls the input stream in much the same way that Control-P controls the output stream. The format for the command is

{*port number*} Control-K

Pressing 0 Control-K directs the Monitor to accept input from the Apple IIc family computer's built-in keyboard.

The Control-P and Control-K commands are the exact equivalents of the BASIC (but not DOS and ProDOS) commands PR# and IN#.

For more direction on how these commands work, see the section "The Standard I/O Links," in Chapter 4.

Hexadecimal arithmetic

You can use the Monitor as a 1-byte hexadecimal addition and subtraction calculator. Just type a line in one of these formats:

{*value*} + {*value*} Return
{*value*} – {*value*} Return

The Apple IIc family member performs the arithmetic and displays the result, as shown in these examples.

```
*20+13
=33
*4A-C
=3E
*
```

Advanced operations

This section describes some ways of using the Monitor commands to speed up your work.

Multiple-command lines

You can put as many Monitor commands on a single line as you like as long as you separate them with spaces and the total number of characters in the line is less than 254. Adjacent single-letter commands such as L, S, I, and N need not be separated by spaces.

You can freely intermix all of the commands except the Store (:) command. Because the Monitor takes all values following a colon and places them in consecutive memory locations, the last value in a Store must be followed by a letter command before another address is encountered. You can use the Normal command as the required letter command in such cases; it usually has no effect and can be used anywhere.

In the following example, you display a range of memory, change it, and display it again, all with one line of commands.

```
*300.307 300:18 69 1 N 300.302
0300- 00 00 00 00 00 00 00 00
0300- 18 69 01
*
```

If the Monitor encounters a character in the input line that it does not recognize as either a hexadecimal digit or a valid command character, it executes all the commands on the input line up to that character, then stops with a beep and ignores the remainder of the input line.

Filling memory

The Move command can be used to replicate a pattern of values throughout a range of memory. To do this, first store the pattern in the first locations in the range

```
*300:11 22 33
*
```

Remember the number of values in the pattern: in this case, it is three. Use the number to compute addresses for the Move command, like this:

{*start+number*} < {*start*} . {*end-number*} M

This Move command first replicates the pattern at the locations immediately following the original pattern, then replicates that pattern following itself, and so on until it fills the entire range.

```
*303<300.32D M
*300.32F
0300- 11 22 33 11 22 33 11 22
0308- 33 11 22 33 11 22 33 11
0310- 22 33 11 22 33 11 22 33
0318- 11 22 33 11 22 33 11 22
0320- 33 11 22 33 11 22 33 11
0328- 22 33 11 22 33 11 22 33
*
```

You can use the Verify command to check whether a pattern repeats itself through memory. This is especially useful to verify that a given range of memory locations all contain the same value. In this example, to see the Verify command detect the discrepancy, you first fill the memory range from $0300 to $0320 with 0's and verify it, then change one location and verify again:

```
*300:0
*301<300.31F M
*301<300.31F V
*304:02
*301<300.31F V
0303-00 (02)
0304-02 (00)
*
```

Repeating commands

You can create a command line that repeats one or more commands over and over. You do this by beginning the part of the command line that you want to repeat with a letter command, such as N, and ending it with the sequence 34:n, where *n* is a hexadecimal number that specifies the position in the line of the command where you want to start repeating; for the first character in the line, *n*=0. The value for *n* must be followed with a space in order for the loop to work properly.

This trick takes advantage of the fact that the Monitor uses an index register to step through the input buffer, starting at location $0200. Each time the Monitor executes a command, it stores the value of the index at location $34; when that command is finished, the Monitor reloads the index register with the value at location $34. By making the last command change the value at location $34, you change this index so that the Monitor picks up the next command character from an earlier point in the buffer.

The only way to stop a loop like this is to press Control-Reset; that is how this example ends.

```
*N 300 302 34:0 N
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
030
*
```

Creating your own commands

The User command (Control-Y) forces the Monitor to jump to memory location $03F8. You can put a JMP instruction there that jumps to your own machine-language program. Your program can then examine the Monitor's registers and pointers or the input buffer itself to obtain its data. For example, here is a program that displays everything on the input line after the Control-Y. The program starts at location $0300; the command line that starts with $03F8 stores a jump to $0300 at location $03F8.

```
*300:A4 34 B9 00 02 20 ED FD C8 C9 8D D0 F5 4C 69 FF
*3F8:4C 00 03
*Control-Y THIS IS A TEST
THIS IS A TEST
*
```

Machine-language programs

The main reason to program in machine language is to get more speed. A program in machine language can run much faster than the same program written in high-level languages such as BASIC or Pascal, but the machine-language version usually takes a lot longer to write. There are other reasons to use machine language: you might want your program to do something that isn't included in your high-level language, or you might just enjoy the challenge of using machine language to work directly on the bits and bytes.

◆ *Note:* If you have never used machine language before, you'll need to learn the 65C02 instructions listed in Appendix A. To become proficient at programming in machine language, you'll have to spend some time at it and study a book on 65C02 programming.

You can get a hexadecimal dump of your program or move it around in memory using the commands described in the previous sections. The Monitor commands in this section are intended specifically for you to use in creating, writing, and debugging machine-language programs.

Running a program

The Monitor command to start execution of your machine-language program is the Go command. When you type an address and press G, the Apple IIc–family computer starts executing machine-language instructions starting at the specified location. If you just press G, execution starts at the last opened location. The Monitor treats this program as a subroutine: it should end with an RTS (return from subroutine) instruction to transfer control back to the Monitor.

The Monitor has some special features that make it easier for you to write and debug machine-language programs, but before you get into that, here is a small machine-language program that you can run using only the simple Monitor commands already described. The program in the example merely displays the letters *A* through *Z*: you store it starting at location $0300, examine it to be sure you typed it correctly, then type `300G` to start it running.

```
*300:A9 C1 20 ED FD 18 69 1 C9 DB D0 F6 60
*300.30C
0300- A9 C1 20 ED FD 18 69 01
0308- C9 DB D0 F6 60
*300G
ABCDEFGHIJKLMNOPQRSTUVWXYZ
*
```

Disassembled programs

Machine-language code in hexadecimal isn't the easiest thing in the world to read and understand. To make this job a little easier, machine-language programs are usually written in assembly language and converted into machine-language code by programs called **assemblers.**

Programs like the Monitor's List command are called **disassemblers.** This command displays machine-language code in assembly-language form. Instead of unformatted hexadecimal gibberish, the List command displays each instruction on a separate line, with a three-letter instruction name, or mnemonic, and a formatted hexadecimal operand. The List command also converts the relative addresses used in branch instructions to absolute addresses.

The Monitor List command has the format

{*location*} L

The List command starts at the specified location and displays as much memory as it takes to make up a screenful (20 lines) of instructions, as shown in the following example:

```
*300 L
0300-    A9 C1        LDA     #$C1
0302-    20 ED FD     JSR     $FDED
0305-    18           CLC
0306-    69 01        ADC     #$01
0308-    C9 DB        CMP     #$DB
030A-    D0 F6        BNE     $0302
030C-    60           RTS
030D-    00           BRK
030E-    00           BRK
030F-    00           BRK
0310-    00           BRK
0311-    00           BRK
0312-    00           BRK
0313-    00           BRK
0314-    00           BRK
0315-    00           BRK
0316-    00           BRK
0317-    00           BRK
0318-    00           BRK
0319-    00           BRK
*
```

The first seven lines of this example are the assembly-language form of the program you typed in the previous example. The rest of the lines are BRK instructions only if this part of memory has 0's in it: other values will be disassembled as other instructions.

The Monitor saves the address that you specify in the List command, but not as the last opened location used by the other commands. Instead, the Monitor saves this address as the program counter, which it uses only to point to locations within programs. Whenever the Monitor performs a List command, it sets the program counter to point to the location immediately following the last location displayed on the screen, so that if you type another List command it displays another screenful of instructions, starting where the previous display left off.

The Step and Trace commands

Step and Trace are Monitor facilities for debugging assembly-language programs. The Step command decodes, displays, and executes one instruction at a time, and the Trace command steps continuously through a program, stopping only when a BRK instruction is executed or the Option key is pressed. You can press the Command key to pause the Trace for approximately one second.

△ Original IIc   The Step and Trace commands are not available in the original Apple IIc. △

Each Step command causes the Monitor to execute the instruction in memory pointed to by the program counter. The instruction is displayed in its disassembled form, then executed. The contents of the 65C02's internal registers are displayed after the instruction is executed. After execution, the program counter is incremented to point to the next instruction in the program.

The format of the Step command is

{*starting location*} S

Here is an example of the Step command, using the following program:

```
$0300:   LDX #02
$0302:   LDA $00,X
$0304:   STA $10,X
$0306:   DEX
$0307:   STA $C030
$030A:   BPL $0302
$030C:   BRK
```

To step through this program, first call the Monitor by typing `CALL -151` and pressing Return, and then from the Monitor type `300S` (to start the Step routine at address $0300). Type `S` to advance each additional step through the program. The Monitor keeps the program counter and the last opened address separate from one another, so you can examine or change the contents of memory while you are stepping through your program. Here's what happens when you step through the program above, examining the contents of location $0012 after the third step. In this example, what you type appears just after the * prompt, and the information on the next two lines—that begin without the * prompt—is what the computer displays on the screen in response.

```
*300S
0300-   A2 02     LDX #02
M=CA A=0A X=02 Y=D8 P=30 S=F8
*S
0302-   B5 00     LDA $00,X
M=CA A=0C X=02 Y=D8 P=30 S=F8
*S
0304-   95 10     STA $10,X
M=CA A=0C X=02 Y=D8 P=30 S=F8
*12
0012- 0C
*S
0306-   CA        DEX
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S
0307-   8D 30 C0 STA $C030
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S
030A-   10 F6     BPL $0302
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S
0302-   B5 00     LDA $00,X
M=CA A=0B X=01 Y=D8 P=30 S=F8
*S
0304-   95 10     STA $10,X
M=CA A=0B X=01 Y=D8 P=30 S=F8
*
```

The Trace command is a continuous version of the Step command; it stops stepping through the program only when you press the Option key or when it encounters a BRK instruction in the program. Press the Command key to pause the trace for one second.

The format of the Trace command is

[*starting location*] T

△ Important Keep the following cautions in mind when using the Step and Trace Monitor commands:

- If the program ends with an RTS instruction, the Trace command repeats indefinitely until stopped with the Option key.
- You can't step or trace through routines that use the same zero-page locations as the Monitor. △

The mini-assembler

Without an assembler, you have to write your machine-language program, take the hexadecimal values for the opcodes and operands, and store them in memory using the Monitor commands described earlier in this chapter.

The mini-assembler lets you enter machine-language programs directly from the keyboard of your Apple. ASCII characters can be entered in mini-assembler programs, exactly as you enter them in the Monitor.

Note that the mini-assembler doesn't accept labels; you must use actual values and addresses.

△ Original IIc The mini-assembler is not available in the original Apple IIc. △

Starting the mini-assembler

To start the mini-assembler, first invoke the Monitor by typing `CALL -151` and pressing Return, and then from the Monitor, type `!` followed by Return. The Monitor prompt character then changes from `*` to `!`.

When you finish using the mini-assembler, press Return from a blank line to return to the Monitor.

To enter code into memory, type the address, a colon, and the instruction. For example, after entering the mini-assembler, you could type

```
!300:STA C030
```

You can enter a series of instructions by typing a space, followed by the instruction, followed by Return:

```
!300:STA C030
! LDA #A0
! INX
```

Each succeeding instruction is placed in the next available memory location. As you type in instructions, each is replaced by the starting address of the instruction, the hexadecimal value(s) of the instruction, followed by mnemonics describing the instruction. For example, the sequence of instructions given above would produce the following on your screen:

```
0300-   8D 30 C0        STA $C030
0303-   A9 A0           LDA #$A0
0305-   E8              INX
```

When you're ready to execute your program, press Return to leave the mini-assembler and return to the Monitor. Monitor commands can't be executed directly from the mini-assembler.

Using the mini-assembler

The mini-assembler saves one address, that of the program counter. Before you start to type a program, you must set the program counter to point to the location where you want the mini-assembler to store your program. Do this by typing the address followed by a colon.

After the colon, type the mnemonic for the first instruction in your program, followed by a space and the operand of the instruction. Now press Return. The mini-assembler converts the line you typed into hexadecimal, stores it in memory beginning at the location of the program counter, and then disassembles it again and displays the disassembled line. It then displays a prompt on the next line.

Now the mini-assembler is ready to accept the second instruction in your program. To tell it that you want the next instruction to follow the first, don't type an address or a colon: just type a space and the next instruction's mnemonic and operand, then press Return. The mini-assembler assembles that line and waits for another.

If the line you type has an error in it, the mini-assembler beeps loudly and displays a caret (^) under or near the offending character in the input line. Most common errors are the result of typographical mistakes: misspelled mnemonics, missing parentheses, and so forth. The mini-assembler also rejects the input line if you forget the space before or after a mnemonic or include an extraneous character in a hexadecimal value or address. If the destination address of a branch instruction is out of the range of the branch (more than 127 locations distant from the address of the instruction), the mini-assembler flags this as an error.

- *Dollar signs:* In this manual, dollar signs ($) in addresses signify that the addresses are in hexadecimal notation. The dollar signs are ignored by the mini-assembler and can be omitted in programs.

```
!300:LDX #02
0300-   A2 02       LDX     #$02
! LDA $00,X
0302-   B5 00       LDA     $00,X
! STA $10,X
0304    95 10       STA     $10,X
! DEX
0306-   CA          DEX
! STA $C030
0307-   8D 30 C0    STA     $C030
! BPL $0302
030A-   10 F6       BPL     $0302
! BRK
030C-   00          BRK
!
```

To leave the mini-assembler and reenter the Monitor, press Return at a blank line.

Your assembly-language program is now stored in memory. You can display it with the List command:

```
*300L
0300-   A2 02        LDX    #$02
0302-   B5 00        LDA    $00,X
0304-   95 10        STA    $10,X
0306-   CA           DEX
0307-   8D 30 C0     STA    $C030
030A-   10 F6        BPL    $0302
030C-   00           BRK
030D-   00           BRK
030E-   00           BRK
030F-   00           BRK
0310-   00           BRK
0311-   00           BRK
0312-   00           BRK
0313-   00           BRK
0314-   00           BRK
0316-   00           BRK
0316-   00           BRK
0317-   00           BRK
0318-   00           BRK
0319-   00           BRK
*
```

Mini-assembler instruction formats

The Apple IIc mini-assembler recognizes 66 mnemonics and 15 addressing formats. The mnemonics are standard, as used in Appendix A, but the addressing formats are somewhat different, as shown in Table 10-1.

An address consists of one or more hexadecimal digits. The mini-assembler interprets addresses the same way the Monitor does: if an address has fewer than four digits, the mini-assembler adds leading 0's; if the address has more than four digits, it uses only the last four.

■ **Table 10-1** Mini-assembler address formats

Addressing mode	Format
Accumulator	*
Implied	*
Immediate	#${*value*}
Absolute	${*address*}
Zero page	${*address*}
Indexed zero page	${*address*},X ${*address*},Y
Indexed absolute	${*address*},X ${*address*},Y
Relative	${*address*}
Indexed indirect	(${*address*},X)
Indirect indexed	(${*address*}),Y
Absolute indirect	(${*address*})

* These instructions have no operands.

There is no syntactical distinction between the absolute and zero-page addressing modes. If you give an instruction to the mini-assembler that can be used in both absolute and zero-page mode, the mini-assembler assembles that instruction in absolute mode if the operand for that instruction is greater than $FF, and it assembles it in zero-page mode if the operand is less than $0100.

Instructions in accumulator mode and implied addressing mode need no operands.

Branch instructions, which use the relative addressing mode, require the target address of the branch. The mini-assembler calculates the relative distance to use in the instruction automatically. If the target address is more than 127 locations distant from the instruction, the mini-assembler sounds a bell (beep), displays a caret (^) under the target address, and does not assemble the line.

If you give the mini-assembler the mnemonic for an instruction and an operand, and the addressing mode of the operand cannot be used with the instruction you entered, then the mini-assembler does not accept the line.

Summary of Monitor commands

Here is a summary of the Monitor commands, showing the syntax diagram for each one.

Examining memory

{*adrs*}Return	Displays the value contained in one location.
{*adrs1*}.{*adrs2*}Return	Displays the values contained in all locations between {*adrs1*} and {*adrs2*}
Return	Displays the values in up to eight locations following the last opened location.
{*adrs*}L	Lists disassembled code starting at {*adrs*} and continuing until the screen is full.

Changing the contents of memory

{*adrs*}:{*val*}{*val*}...	Store command. Stores the values in consecutive memory locations starting at {*adrs*}.
:{*val*}{*val*}...	Stores values in memory starting at the next changeable location.

Moving and comparing

{*dest*}<{*start*}.{*end*}M	Move command. Copies the values in the range {*start*}.{*end*} into the range beginning at {*dest*}.
{*dest*}<{*start*}.{*end*}V	Verify command. Compares the values in the range {*start*}.{*end*} to those in the range beginning at {*dest*}.

The Register command

Control-E	Examine command. Displays the locations where the contents of the 65C02's registers are stored and opens them for changing.

Miscellaneous Monitor commands

I	Inverse command. Sets inverse display mode.
N	Normal command. Sets normal display mode.
Control-B	BASIC command. Enters the language currently active (normally Applesoft).
Control-C	Continue BASIC command. Returns to the language currently active (normally Applesoft).
{*val*}+{*val*}	Adds the two values and prints the hexadecimal result.
{*val*}–{*val*}	Subtracts the second value from the first and prints the result.
{*port*}Control-P	Redirects output to the device connected to port number {*port*}. If {*port*}=0, sends output to the video display. Use only when the enhanced video firmware is not active (checkerboard cursor).
Esc Control-Q	Redirects output to video display when enhanced video firmware is active (solid block cursor).
{*port*}Control-K	Takes input from the device connected to port number {*port*}. If {*port*}=0, accepts input from the keyboard.
Control-Y	User command. Jumps to the machine-language subroutine at location $03F8.

Running and listing programs

{*adrs*}G	Go command. Transfers control to the machine-language program beginning at {*adrs*}.
{*adrs*}L	List command. Disassembles and displays 20 instructions starting at {*adrs*}. Subsequent L's display 20 more instructions each.
{adrs}S	Step command. Steps through the program one instruction at a time.
{adrs}T	Trace command. Traces continously thorugh the program.

Chapter 11 **Hardware**

Most of this manual describes functions—what the Apple IIc does. This chapter, on the other hand, describes objects—the pieces of hardware the Apple IIc family members use to carry out their functions. If you are designing a device to connect to the Apple IIc back panel, or if you just want to know more about how the Apple IIc or Apple IIc Plus is built, you should study this chapter. ■

Environmental specifications

The Apple IIc family members are quite sturdy when used in the way they were intended: as transportable computers, made for use in an indoor environment. You can carry one by its handle from room to room, but for longer trips you should use its carrying case or some other protective container (such as an attaché case). Table 11-1 defines the conditions under which Apple IIc family members are designed to function properly.

■ **Table 11-1** Environmental specifications

Operating temperature	10° to 40° C (50° to 104° F)
Relative humidity	20% to 95%
Line voltage	105 to 129 VAC (normal USA voltage range)

You should treat an Apple IIc or Apple IIc Plus with the same kind of care as any other electrical appliance; protect it from physical abuse, and be careful not to bump it against furniture when you move it around. Put it in an attache case or other protective covering if you carry it outside. You should also protect the mechanical keyboard and the electrical connectors inside the case from spilled liquids, particularly those with dissolved contaminants, such as soups, fruit juices, and carbonated soft drinks.

In normal operation (with the handle locked in its down position), enough air flows through the openings in the case to keep the insides from getting too hot. If you do overheat your Apple IIc or Apple IIc Plus computer—for example, by blocking the upper or lower ventilation openings—the first symptom will be erratic operation, such as unexpectedly changed data. (The memory devices are especially sensitive to heat.) Letting the machine cool down by turning it off for a while and unblocking the vents before using it again will bring it back to normal operation. The only exception to this is if you have gotten your computer *too* hot and physically damaged some internal component.

Disks are another heat-sensitive element of the system. If the built-in drive becomes too hot, a disk within can warp or even melt. A melted or warped disk can't be used again.

Block diagrams

Figure 11-1 shows a block diagram for the Apple IIc computers (original, UniDisk 3.5, and memory-expansion versions). Figure 11-2 shows a block diagram for the Apple IIc Plus computer. The Apple IIc computers include the following components:

- The central processing unit (CPU), a 65C02 microprocessor, which executes program instructions to transfer data to and from memory, perform calculations, and otherwise manipulate data.
- Read-only memory (ROM), which contains the system firmware used for startup code, disk drivers, the System Monitor, and a variety of other programs.
- Random-access memory (RAM), used for stack, zero page, operating systems, application programs, and data. There is more RAM than can be addressed at one time by the CPU, so it is divided into two parts: main RAM and auxiliary RAM.
- The video circuits, which can address RAM directly and which send signals to the video connector and the video expansion connector. The video connector is used for monochrome and color monitors, and the video expansion connector is used for other devices such as LCD displays and RF modulators (for TV sets).
- The memory management unit (MMU), which contains most of the logic that controls memory addressing.
- The Video latch and 80 latch, which control whether main RAM or auxiliary RAM data is put on the video bus.
- The 80DIR bidirectional bus transceiver, which—together with the Video latch and 80 latch—controls which data (main RAM or auxiliary RAM) is put onto the video data bus and which is put onto the processor bus.
- The timing generator (TMG), which uses the signal from a 14 MHz oscillator to generate all the clock and timing signals used by the system.
- The general logic unit (GLU), which performs a variety of logic functions.
- The input/output unit (IOU), which implements several soft switches, detects mouse movement, and generates timing and control signals for the video circuits, the speaker, and the keyboard.
- The audio circuit (AUD), which generates an analog signal for the speaker and output jack.

- The keyboard decoder, which translates a key position into the appropriate character code and puts it on the data bus.
- The MUX chip, which reads data from the mouse, keyboard, and game paddles, and converts it into serial data for the CPU.
- Two asynchronous communication interface adapters (ACIAs), which provide the control signals and parallel/serial conversion for the two serial ports. The ACIAs are supported by line drivers and receivers.
- The disk controller unit, also known as the Integrated Woz Machine (IWM), which provides the interface for the built-in and external disk drives.
- An internal speaker and external audio jack
- An external connectors for floppy disks
- An external connector for a mouse or hand control
- An external video connector
- An external video expansion connector
- Two external connectors for serial devices
- An internal connector for a 5.25-inch floppy disk
- An internal connector for a memory expansion card (memory expansion Apple IIc only)

■ **Figure 11-1** Apple IIc block diagram

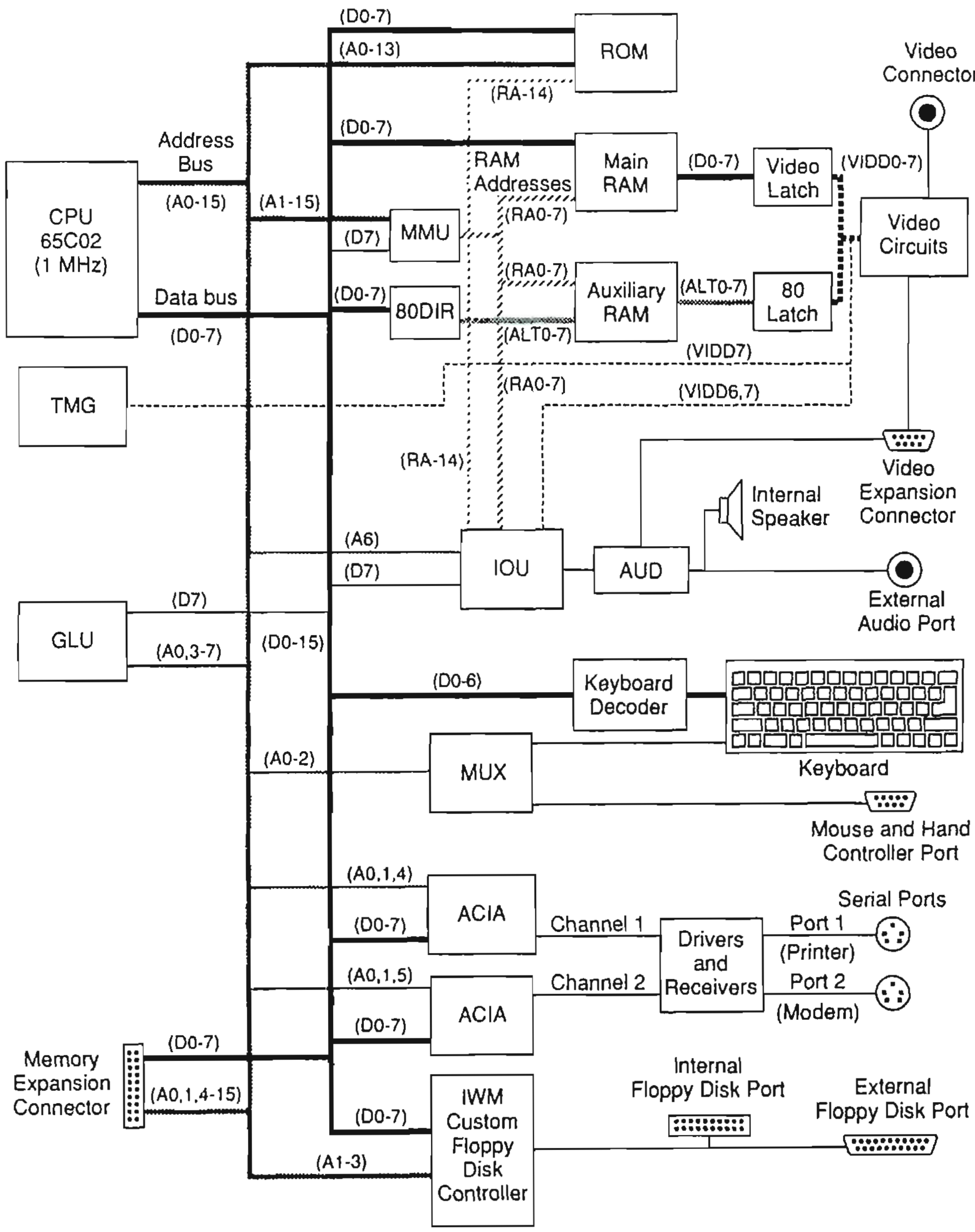
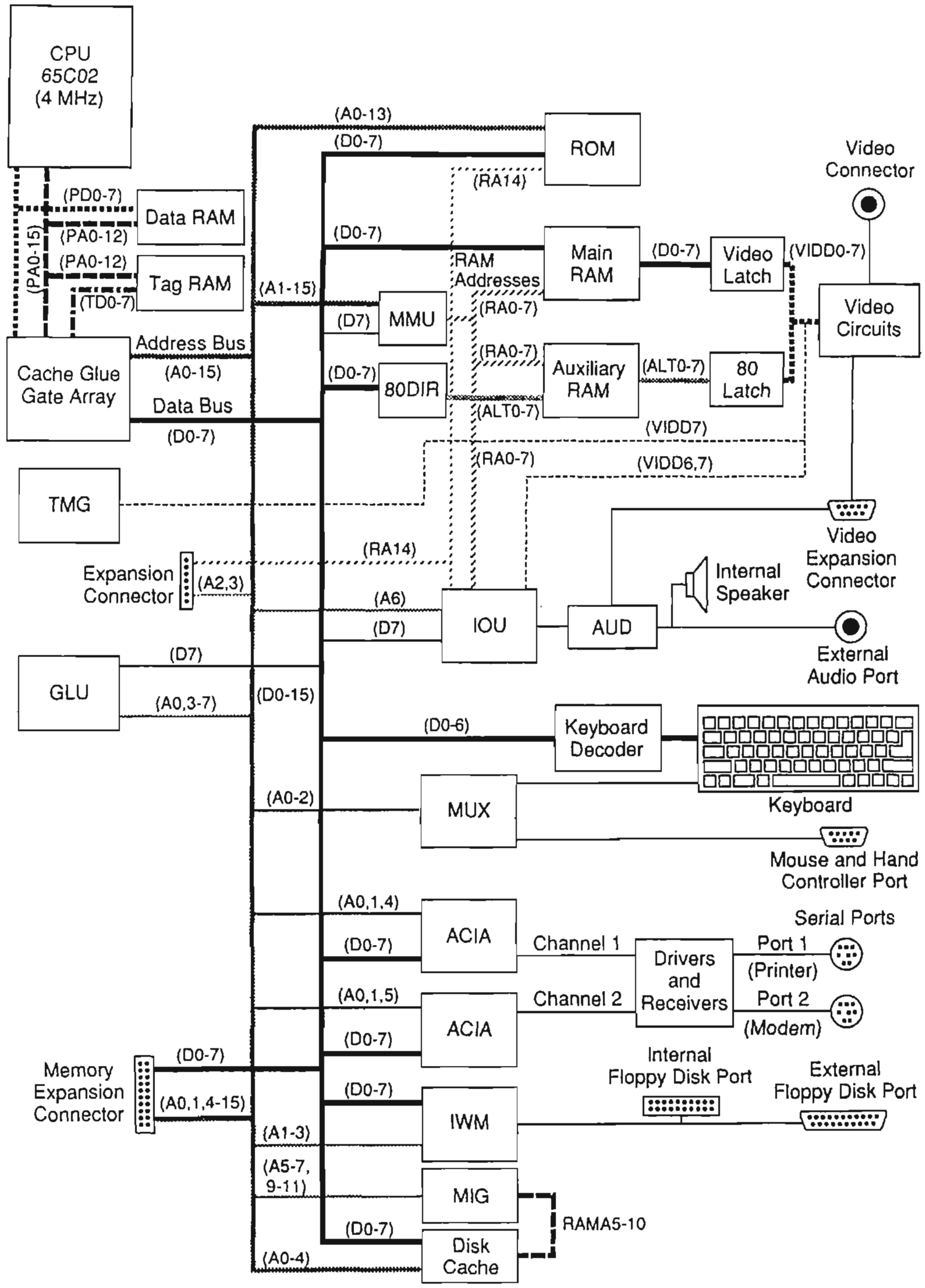


Figure 11-2 shows a block diagram for the Apple IIc Plus computer. The Apple IIc Plus differs from the Apple IIc primarily in that the Apple IIc Plus has a 65C02 that can operate at 4 MHz, as well as two new custom ICs, several supporting ICs, and two new internal connectors, as follows:

- The 84-pin cache glue gate array (CGGA), which implements a RAM cache for the CPU and provides the clock for the CPU
- A 16 MHz crystal oscillator
- Two 8 KB static RAMs for the RAM cache
- The multidrive interface glue chip (MIG), which implements a RAM buffer and provides control signals for internal and external 3.5-inch disk drives
- A 2 KB static RAM for the 3.5-inch disk drive RAM buffer
- An internal modem connector
- An internal expansion connector, to supply additional signals for internal memory expansion cards

Notice that in the Apple IIc Plus computer, the system address and data buses do not connect directly to the 65C02; rather, the CGGA interfaces the processor data and address buses to the system data and address buses. This interface allows the 65C02 to run at 4 MHz while the system data and address buses run at 1 MHz. The CGGA also controls the RAM cache, which further increases the efficiency and overall speed of processor operation in the Apple IIc Plus. The Apple IIc Plus has no external sound jack.

■ **Figure 11-2** Apple IIc Plus block diagram



Power supplies

The electronic hardware of the Apple IIc family operates on DC voltages. These voltages are produced from standard household AC voltages (105–129 VAC). The AC voltage is called the *line voltage*, and the DC voltages are called *supply voltages*.

The original, UniDisk 3.5, and memory expansion versions of the Apple IIc use an external transformer to rectify the AC line voltage into a 15 VDC input to the computer. This voltage is then converted by an internal voltage converter to the various supply voltages required by the main logic board, disk drive, and external devices. The main supply voltage is connected to the internal converter circuits by a power supply cord connected at the back panel.

The Apple IIc Plus uses a completely internal, modular power supply. Line voltage is connected to the AC power input port at the back panel.

If you purchased your Apple IIc outside the USA, consult Appendix E for its power supply characteristics.

The Apple IIc power supply

The original, UniDisk 3.5, and memory expansion versions of the Apple IIc use an external power transformer and an internal voltage converter.

The Apple IIc external transformer

The transformer is a standard 1:8 step-down AC line transformer with a voltage rectifier circuit. AC line voltage is received on the primary side and stepped down to the secondary side. The stepped down AC voltage is then rectified and sent to the Apple IIc internal voltage converter through the power supply cord. The main supply voltage output is from 9 to 20 VDC.

The specifications of the external transformer are listed in Table 11-2. The line voltage must be in the range given in Table 11-2.

▲ **Warning** The supply cord of the Apple IIc external transformer must be plugged into a three-wire 115-volt (nominal) outlet. A two-wire outlet is not properly grounded—using it can result in damage to the external transformer and perhaps to the Apple IIc as well.

If you are unable to insert the plug into the outlet, contact a licensed electrician to replace the outlet and, if necessary, install a grounding conductor.

Do not defeat the purpose of the grounding plug. ▲

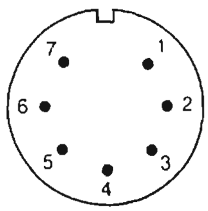
■ **Table 11-2** Apple IIc power transformer specifications

Line voltage	105 to 129 VAC, 60 Hz
Maximum power consumption	25 W
Supply voltage	+15 VDC (nominal)
Supply current	1.2 A (nominal)

The external power connector

The power supply cable is connected to the internal converter by means of a 7-pin DIN connector on the back panel. The connector pins are identified in Figure 11-3 and Table 11-3.

■ **Figure 11-3** Apple IIc external power connector



■ **Table 11-3** Apple IIc external power connector signals

Pin	Signal	Description
1, 7		Not connected
2, 3	Ground	Common electrical ground
4	Chassis	Chassis ground
5, 6	+15V	+15-volt DC input to converter

The Apple IIc internal voltage converter

The internal voltage converter operates from the main supply voltage provided by the external power transformer. It provides DC supply voltages for all internal components and up to three external disk drives. The internal converter specifications are listed in Table 11-4. Negative 5 volts (–5 VDC) is derived on the main logic board itself from the –12 VDC supply .

■ **Table 11-4** Apple IIc internal voltage converter specifications

Input voltage	+9 to 20 VDC	
Maximum power consumption	25 W	
Supply voltages	+5V ±5%	
	+12V ±10%	
	–12V ±10%	
Maximum supply currents	+5V:	1.5 A
	+12V:	0.6 A continuous
		0.9 A intermittent
		1.5 A surge (for < 100 ms)
	–12V:	0.1 A
	(–5V:	50 mA)
Maximum case temperature	60° C	(140° F)

The internal converter—called a *switching-type converter*—works by using the DC voltage input to power a variable-frequency oscillator. The oscillator drives a small transformer with several separate windings to produce the various supply voltages. A circuit compares the voltage of the +5-volt supply with a reference voltage and feeds an error signal back to the oscillator circuit. The oscillator circuit uses the error signal to control the duty cycle of its oscillation and keep the supply voltages in their normal ranges.

The converter includes circuitry to protect itself and the other electronic parts of the Apple IIc by limiting all three supply voltages whenever it detects one of the following malfunctions:

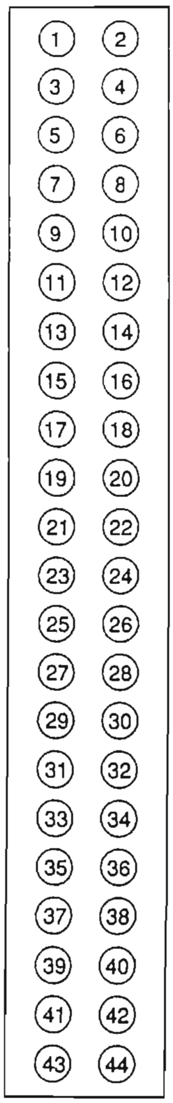
- any supply voltage short-circuited to ground
- any supply voltage outside the normal range

Whenever one of these malfunctions occurs, the protection circuit varies the duty cycle of the oscillator, dropping all supply voltages to 0 if they cannot be brought back into their normal range.

The Apple IIc internal power connector

The internal converter is connected to the main logic board by means of a 44-pin card edge connector. The connector pins are identified in Figure 11-4 and Table 11-5.

■ **Figure 11-4** Apple IIc internal power connector



■ **Table 11-5** Apple IIc internal power connector signals

Pin	Signal	Description
1–5	GND	Ground
6	–12V	–12-volt DC supply
7	GND	Ground
8	–12V	–12-volt DC supply
9	GND	Ground
10	N.C.	Not connected
11	GND	Ground
12	FFF	Switch ground/12V in
13	GND	Ground
14	+12F IN	+12V DC power in
15	GND	Ground
16	+12F IN	+12V DC power in
17	GND	Ground
18	+12F IN	+12V DC power in
19	GND	Ground
20	+12F IN	+12V DC power in
21	GND	Ground
22	+12F IN	+12V DC power in
23–27	GND	Ground
28	+5V	+5V supply
29	GND	Ground
30	+5V	+5V supply
31	GND	Ground
32	+5V	+5V supply
33	GND	Ground
34	+5V	+5V supply
35	GND	Ground
36	+12V	+12V supply
37	GND	Ground
38	+12V	+12V supply
39	GND	Ground
40	+12V	+12V supply
41–44	GND	Ground

The Apple IIc Plus internal power supply

The Apple IIc Plus uses an internal switching-type power supply. Normal household AC line voltage is received on the primary side and stepped down and rectified to +5V, +12V, and –12 VDC on the secondary side. The power supply provides DC supply voltages for all internal components and up to three external disk drives. Negative 5 volts (–5 VDC) is derived on the main logic board itself from the –12 VDC supply by an R-C network. The basic specifications of the internal power supply are listed in Table 11-6. The line voltage must be in the range given in Table 11-6.

▲ **Warning** The power supply cord of the Apple IIc Plus must be plugged into a three-wire 115-volt (nominal) outlet. A two-wire outlet is not properly grounded—using it can result in damage to the power supply and perhaps to other components as well.

If you are unable to insert the plug into the outlet, contact a licensed electrician to replace the outlet and, if necessary, install a grounding conductor.

Do not defeat the purpose of the grounding plug. ▲

■ **Table 11-6** Apple IIc Plus internal power supply specifications

Line voltage	105 to 129 VAC, 60 Hz	
Maximum power output	20 W	
Supply voltages	+5V ±5%	
	+12V ±10%	
	–12V ±10%	
Maximum supply currents	+5V:	1.5 A
	+12V:	0.6 A continuous
		0.9 A intermittent
		1.5 A surge (for < 100 ms)
	–12V:	0.1 A
	(–5V:	50 mA)
Maximum case temperature	60° C	(140° F)

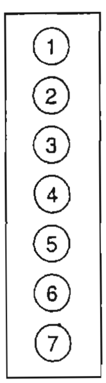
The power supply includes circuitry to protect itself and the other electronic parts of the Apple IIc Plus by limiting all three supply voltages whenever it detects one of the following malfunctions:

- any supply voltage short-circuited to ground
- any supply voltage outside the normal range

Whenever one of these malfunctions occurs, the protection circuit varies the duty cycle of the oscillator, droping all supply voltages to 0 if they cannot be brought back into their normal range.

The internal power supply is connected to the main logic board by a 7-pin connector. The connector pins are shown in Figure 11-5 and Table 11-7. Notice that this connector differs from the one used in earlier Apple IIc computers, shown in Figure 11-4.

■ **Figure 11-5** Apple IIc Plus internal power connector



■ **Table 11-7** Apple IIc Plus internal power connector signals

Pin	Signal	Description
1	–12V	–12-volt supply
2	+12V	+12-volt supply
3	+5V	+5-volt supply
4	+5V	+5-volt supply
5–7	GND	Ground

The 65C02 microprocessor

The Apple IIc family uses a CMOS 6502 (designated as 65C02) microprocessor as its central processing unit (CPU). The 65C02 in the Apple IIc runs at a clock rate of 1.023 MHz and performs up to 500,000 8-bit operations per second. The 65C02 in the Apple IIc Plus runs at a clock rate of either 1.023 MHz or 4 MHz and performs up to 2,000,000 8-bit operations per second.

◆ *Note:* The clock rate is not a very good criterion for comparing different types of microprocessors. The 65C02 has a simpler instruction cycle than most other microprocessors and it uses instruction pipelining for faster processing. The speed of the 65C02 with a 1 MHz clock is equivalent to many other types of microprocessors with clock rates up to 5 MHz.

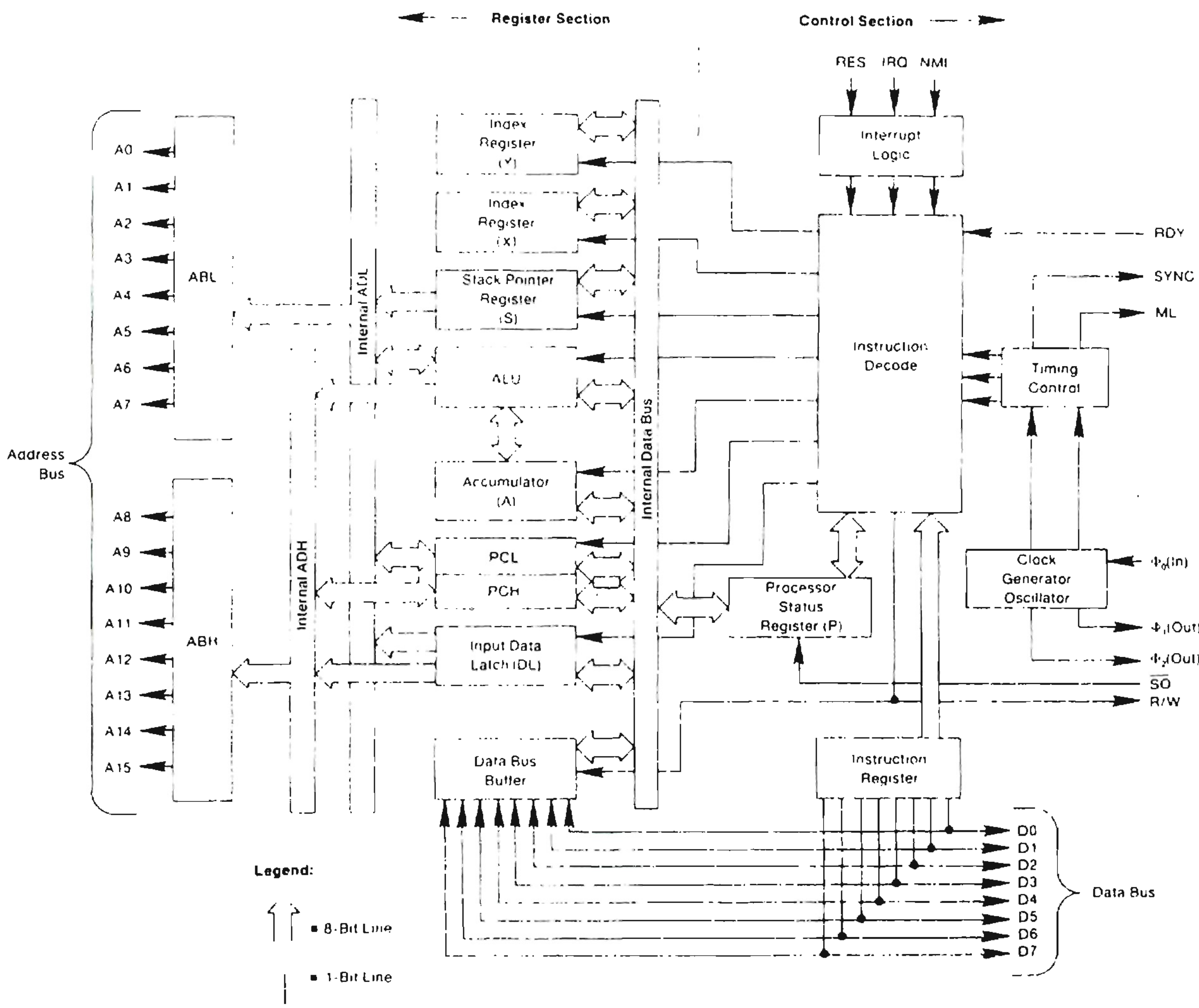
In addition to requiring less power than earlier 6502 processors, the 65C02 in the Apple IIc family has 27 new instructions. However, programs that use these additional instructions are not backward compatible with other Apple II computers not equipped with a 65C02.

The manufacturer's data sheet for the 65C02 microprocessor is reproduced in Appendix A.

65C02 block diagram

Figure 11-6 is a block diagram of the 65C02 microprocessor. Table 11-8 contains the general specifications of this chip. The 65C02 has a 16-bit address bus, giving it an address space of 64 KB. The Apple IIc family uses special techniques to address a total of more than 64 KB (see Chapter 2).

■ **Figure 11-6** 65C02 block diagram; (copyright © 1982 by NCR Corporation; used by permission)



■ **Table 11-8** 65C02 microprocessor specifications

Type	65C02
Register complement	8-bit accumulator (A) 8-bit index registers (X,Y) 8-bit stack pointer (S) 8-bit Processor Status (P) 16-bit program counter (PC)
Data bus	8 bits wide
Address bus	16 bits wide
Address range	65,536 (64 K)
Interrupts	IRQ signal (maskable) NMI signal (nonmaskable) BRK instruction (programmed)
Operating voltage	+5V (±5%)
Power dissipation	5 mW (at 1 MHz)

65C02 timing

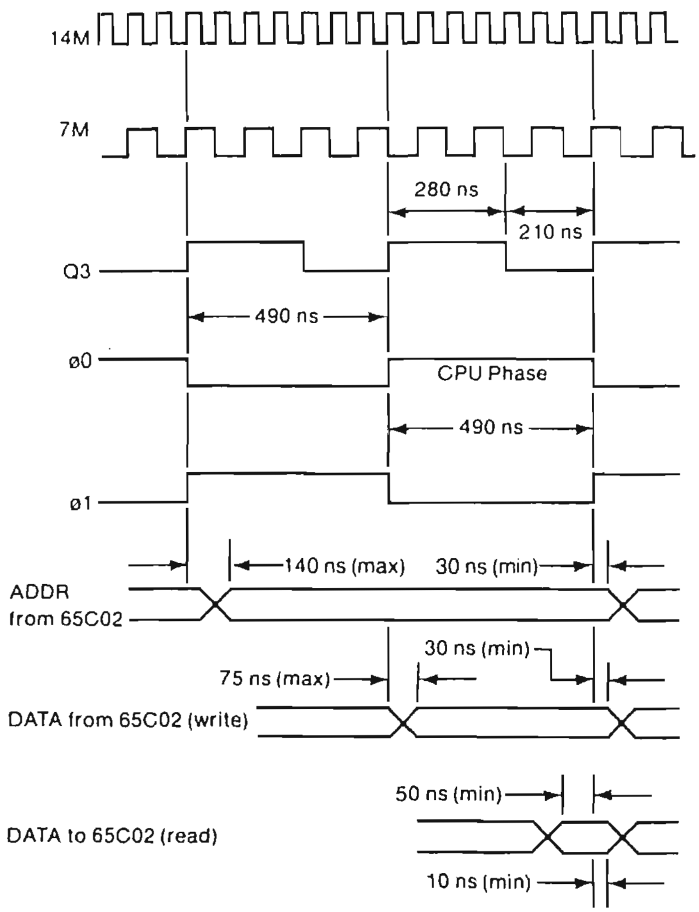
The operation of the Apple IIc and Apple IIc Plus computers is controlled by a set of synchronous timing signals, sometimes called ***clock signals***. The Apple IIc and Apple IIc Plus use a 14.318 MHz master timing signal, called *14M*, to produce all the other timing signals. These timing signals perform two major tasks: controlling the computing functions, and generating the video display. In the Apple IIc family computers, the system timing signals PH0 and PH1 are generated by the timing generator IC, the TMG. The timing signals directly involved with the 65C02's operation are described in this section. Other timing signals are described in the section "The Timing Generator (TMG)," later in this chapter.

The relationships of the main system timing signals are diagrammed in Figure 11-7, and the signals are listed in Table 11-9. The system clock signals PH0 and PH1 are complementary signals at a frequency of 1.023 MHz. The PH0 signal is also connected to the clock input of the 65C02. Because the signal PH0 is used in the Apple IIc family both as the 65C02 clock input and as the system clock, it is a bit early compared to the ø2 system clock described in Appendix A.

△ Apple IIc Plus In the Apple IIc Plus computer, the PH0 clock signal is input to the CGGA, and the CGGA provides a separate clock to the 65C02. This arrangement allows the CGGA to operate the 65C02 at either 1.023 MHz or 4 MHz, as required. In addition, the 65C02 processor's address and data lines are connected to the CGGA; the CGGA provides the address and data signals for the rest of the system. In other words, the CGGA and the 65C02 together effectively constitute the CPU of the Apple IIc Plus computer.

When the term *CPU* is used in the discussion of timing signals, you should understand it to mean the 65C02 processor in the Apple IIc, and the combination of the CGGA and the 65C02 in the Apple IIc Plus. The CGGA is described in the section "The Apple IIc Plus Cache Glue Gate Array (CGGA)," later in this chapter. △

■ **Figure 11-7** System timing signals



■ **Table 11-9** System timing signal descriptions

Signal	Description
14M	Master oscillator, 14.318 MHz; also 80-column dot clock
VID7M	Intermediate timing signal and 40-column dot clock
Q3	Intermediate timing signal, 2.045 MHz with asymmetrical duty cycle
PH0	Phase 0 of system clock, 1.023 MHz; complement of PH1; clock input to CPU
PH1	Phase 1 of system clock, 1.023 MHz; complement of PH0

The CPU's operations are related to the clock signals in a simple way: internal during PH1, external during PH0. The CPU puts an address on the address bus during PH1. This address is valid not later than 110 nanoseconds after PH1 goes high and remains valid through all of PH0. The CPU reads or writes data during PH0. If the CPU is writing, the read/write signal is low during PH0 and the CPU puts data on the data bus. The data are valid not later than 75 nanoseconds after PH0 goes high. If the CPU is reading, the read/write signal remains high. Data on the data bus must be valid no later than 50 nanoseconds before the end of PH0.

More information about the 65C02 and its instruction set is in Appendix A.

The custom integrated circuits

Most of the circuitry that controls memory and I/O addressing in the Apple IIc family computers is in five custom integrated circuits:

- the memory management unit (MMU)
- the input-output unit (IOU)
- the timing generator (TMG)
- the general logic unit (GLU)
- the disk controller unit, also known as the Integrated Woz Machine (IWM)

The soft switches that control the various I/O and addressing modes of the Apple IIc family are addressable flags inside the MMU, IOU, and GLU. The functions of the MMU and IOU are not as independent as their names suggest; working together, they generate all the addressing signals. For example, the MMU generates the RAM address signals for the CPU, while the IOU generates similar RAM address signals for the video display and most I/O hardware addresses.

The Apple IIc Plus computer has two additional integrated circuit chips:

- the multidrive interface glue (MIG)
- the cache glue gate array (CGGA)

The MIG provides interface logic for disk drives. The CGGA allows the Apple IIc Plus to operate at a much faster speed than the other Apple IIc computers.

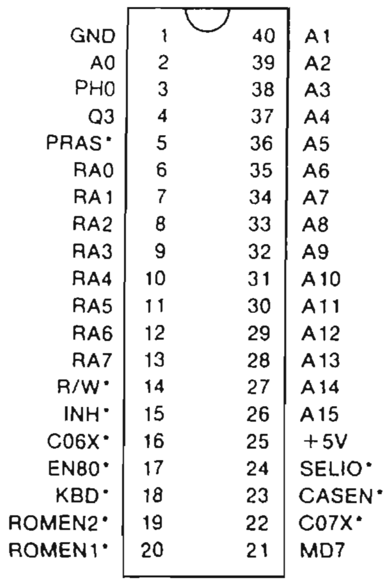
The memory management unit (MMU)

The circuitry inside the MMU implements these soft switches:

- Page 2 display (Page2) (described in Chapter 6)
- high-resolution mode (HiRes) (Chapter 6)
- store to 80-column display (80Store) (Chapter 6)
- select bank 2 (Bank2) (Chapter 2)
- enable bank-switched RAM (EnlCRAM) (Chapter 2)
- read auxiliary memory (RAMRd) (Chapter 2)
- write auxiliary memory (RAMWrt) (Chapter 2)
- auxiliary stack and zero page (AltZP) (Chapter 2)
- reset mouse Y interrupt (RstYInt) (Chapter 9)
- reset mouse X interrupt (RstXInt) (Chapter 9)

These switches are available on MMU pin 21, which is connected to bit 7 on the data bus. Figure 11-8 shows the MMU pinouts; Table 11-10 describes the signals.

■ **Figure 11-8** MMU pinouts



△ Important A signal name followed by an asterisk (PRAS*) is active low—that is, it is asserted when the signal is at a TTL low (0V) level. △

The 64 KB dynamic RAMs used in the Apple IIc family use a multiplexed address, as described later in this chapter. The MMU generates this multiplexed address for memory reading and writing by the 65C02 CPU.

■ **Table 11-10** MMU signal descriptions

Pin	Signal	Description
1	GND	Power and signal common
2	A0	System address input
3	PH0	System clock, phase 0, input
4	Q3	Timing signal input
5	PRAS*	Memory row-address strobe
6–13	RA0–RA7	Multiplexed address output
14	R/W*	System read-write control signal
15	INH*	Inhibits main memory use (tied high through a pullup resistor)

■ **Table 11-10** MMU signal descriptions (continued)

Pin	Signal	Description
16	C06X*	Causes $C06x outputs to go to 0 during PH0
17	EN80*	Enables auxiliary RAM
18	KBD*	Enables keyboard data bits 0-6
19	ROMEN2*	Enables ROM (tied to ROMEN1*)
20	ROMEN1*	Enables ROM (tied to ROMEN2*)
21	MD7	State of MMU flags on data bus bit 7
22	C07X*	Causes $C07x outputs to go to 0 during PH0
23	CASEN*	Enables main RAM
24	SELIO*	Goes to 0 during PH0 for any access to $C0 page except $C08x, Bx, Cx, or Fx
25	+5V	Power
26-40	A15-A1	System address input

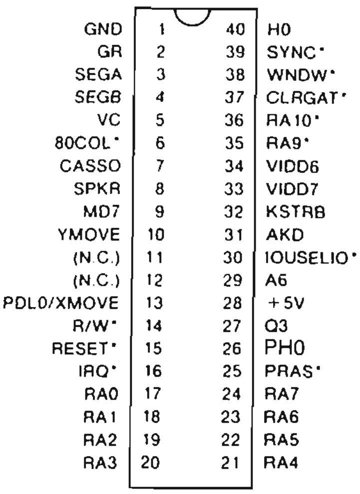
The input/output unit (IOU)

Input/output unit (IOU) implements the soft switches and status bits that control the following features:

- Page 2 display (Page2) (described in Chapter 2)
- high-resolution mode (HiRes) (Chapter 2)
- text mode (TEXT) (Chapter 6)
- mixed mode (MIXED) (Chapter 6)
- 80-column display (80Col) (Chapter 6)
- character-set select (AltChar) (Chapter 6)
- any-key-down flag (AKD) (Chapter 5)
- keyboard stroke (Chapter 5)
- mouse movement interrupts (X0, Y0) (Chapter 9)
- vertical blanking interrupt (VBlInt) (Chapter 9)

These switches are available on IOU pin 9, which is connected to bit 7 on the data bus. Figure 11-9 shows the IOU pinouts; Table 11-11 describes the signals.

■ **Figure 11-9** IOU pinouts



The dynamic RAMs used in the Apple IIc family require a multiplexed address, as described in the section "Dynamic RAM Refreshment" later in this chapter. The IOU generates this multiplexed address during clock phase 1 for the data transfers required for display and memory refresh. The way this address is generated is described under "The Video Counters," later in this chapter.

■ **Table 11-11** IOU signal descriptions

Pin	Signal	Description
1	GND	Power and signal common
2	GR	Graphics mode enable
3	SEGA	In text mode, works with VC (see pin 5) and SEGB to determine character row address
4	SEGB	In text mode, works with VC (see pin 5) and SEGA; in graphics mode, selects Hi-Res when low, Lo-Res when high
5	VC	Vertical counter bit: in text mode, SEGA, SEGB, and VC determine which of the eight rows of a character's dot pattern to display; in Lo-Res, selects upper or lower block defined by a byte
6	80COL*	80-column video enable

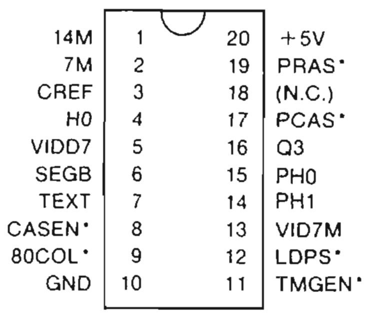
■ **Table 11-11** IOU signal descriptions (continued)

Pin	Signal	Description
7	RA14	ROM address 14, used for ROM bank switching; derived from Cassette Out signal (CASSO)
8	SPKR	Speaker output signal
9	MD7	Internal IOU flags for data bus (bit 7)
10	YMOVE	Detects mouse movement along Y axis
11	N.C.	Not used
12	N.C.	Not used
13	PDL0/XMOVE	Detects mouse movement along X axis
14	R/W•	65C02 read-write control signal
15	RESET•	Power on and reset output
16	IRQ•	Maskable interrupt line to 65C02
17-24	RA0–RA7	Video refresh multiplexed RAM address (phase 1)
25	PRAS•	Row-address strobe (phase 0)
26	PH0	System clock, phase 0
27	Q3	Intermediate timing signal
28	+5V	Power
29	A6	System address bit 6
30	IOUSELIO•	*Derived from the SELIO• output for MMU pin 24*
31	AKD	Any-key-down signal
32	KSTRB	Keyboard strobe signal
33,34	VIDD7,VIDD6	Video display data bits
35,36	RA9•,RA10•	Video display control bits
37	CLRGAT•	Color-burst gate (enable)
38	WNDW•	Blanking signal for display
39	SYNC•	Synchronization signal for display
40	H0	Horizontal timing signal for display (low bit of character counter)

The timing generator (TMG)

A custom timing generator chip (TMG) generates several timing and control signals in the Apple IIc family computers. The TMG pinouts are shown in Figure 11-10; the signals are listed in Table 11-12. For more information on system timing signals, see the section "65C02 Timing," earlier in this chapter.

■ **Figure 11-10** TMG pinouts



■ **Table 11-12** TMG signal descriptions

Pin	Signal	Description
1	14M	14.318 MHz master timing signal input
2	7M	7.159 MHz timing signal
3	CREF	3.5795 MHz color reference timing signal
4	H0	Horizontal video timing signal
5	VIDD7	Video data bit 7
6	SEGB	Video timing signal
7	TEXT	Video display text-modes enable
8	CASEN*	RAM enable (CAS enable)
9	80COL*	Enables 80-column display mode
10	GND	Power and signal common
11	TMGEN*	Enables master timing
12	LDPS*	Video shift-register load enable
13	VID7M	Video dot clock enable, 7 MHz or continuous 0
14	PH1	System clock, phase 1
15	PH0	System clock, phase 0
16	Q3	Intermediate timing and strobe signal
17	PCAS*	RAM column-address strobe
18	N.C.	Reserved for testing
19	PRAS*	RAM row-address strobe
20	+5V	Power

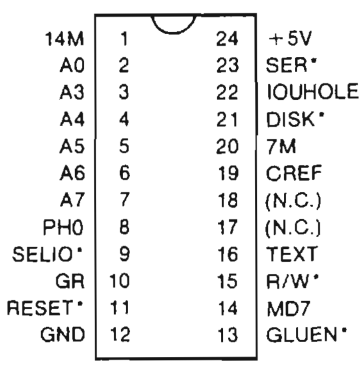
The general logic unit (GLU)

The general logic unit is a single chip that contains the miscellaneous logic required for the system. It provides

- all RAM read/write timing
- Double Hi-Res enable/disable
- soft-switch status registers
- write command registers
- IOU control for mouse interrupts
- Double Hi-Res soft switches

The GLU's pin assignments are shown in Figure 11-11 and its signals are listed in Table 11-13.

■ **Figure 11-11** GLU pinouts



■ **Table 11-13** GLU signal descriptions

Pin	Signal	Description
1	14M	Master clock (14.318 MHz)
2,3–7	A0,A3–A7	Address lines to select least significant byte of addresses on C0 page
8	PH0	System clock, phase 0
9	SELIO*	Device select for selecting most significant byte of the address
10	GR	Graphics mode select line

■ **Table 11-13** GLU signal descriptions (continued)

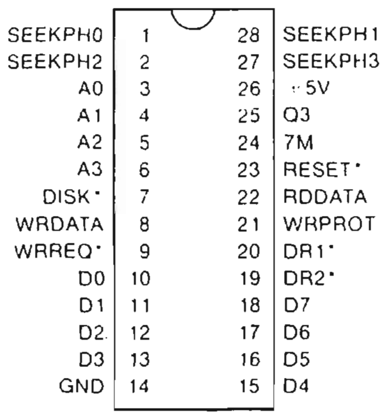
Pin	Signal	Description
11	RESET*	Master reset for system; resets GLU
12	GND	Ground reference and negative supply
13	GLUEN*	Enables GLU
14	MD7	Indicates status of MMU flags on data bus bit 7
15	R/W*	Read/write qualifier input from processor
16	TEXT	Video text signal from TMG; set to inverse of GR, except in Double Hi-Res mode
17,18	N.C.	Not used
19	CREF	Color reference signal
20	7M	7 MHz clock output
21	DISK*	Disk controller device select output
22	IOUHOLE	Controls IOUSELIO
23	SER*	Serial controller device select output
24	+5V	+5 volt supply

The disk controller unit (IWM)

The IWM is a disk controller chip that contains all of the circuitry necessary to provide a complete electrical interface for the 5.25-inch disk drives.

Right after reset, the IWM is configured as an integrated group code recording (GCR) disk drive controller. It also has a status register, mode register, and multiple operating modes. It provides both synchronous and asynchronous modes, as well as a fast mode with a data rate twice that of normal disk I/O speeds. Figure 11-12 shows the IWM pin assignments; Table 11-14 describes the IWM signals.

■ **Figure 11-12** IWM pinouts



■ **Table 11-14** IWM signal descriptions

Pin	Signal	Description
1	SEEKPH0	Disk drive register select line CA0, one of four programmable output lines
2	SEEKPH2	Disk drive register select line CA2
3	A0	The data input to the state bit selected by A1 to A3
4–6	A1–A3	These three inputs select one of the 8 bits in the state register to be updated
7	DISK*	Device enable; the falling edge of DISK* latches information on A1 to A3. The rising edge of either Q3 or DISK* qualifies write register data
8	WRDATA	The serial data output; each 1 bit causes a transition on this output
9	WRREQ*	Programmable buffered output line
10–13	D0–D3	D0 to D7 make up the bidirectional data bus
14	GND	Ground reference and negative supply
15–18	D4–D7	The remaining bits of the bidirectional data bus
19	DR2*	Drive 2 select
20	DR1*	Drive 1 select
21	WRPROT	Write-protect input; this can be polled via bit 7 of the status register
22	RDDATA	Serial data input line; the IWM synchronizes the falling transition of each pulse

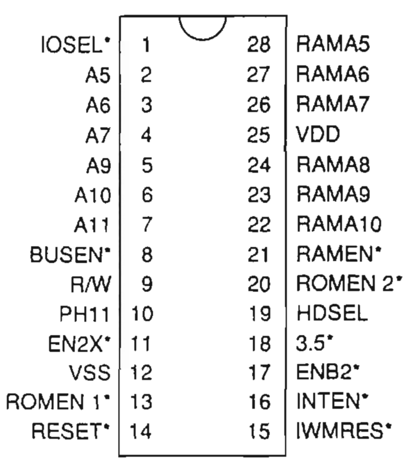
■ **Table 11-14** IWM signal descriptions (continued)

Pin	Signal	Description
23	RESET*	IWM reset: places all IWM outputs in their inactive state and sets all state and mode register bits to 0
24	7M	7 MHz clock input
25	Q3	A 2.0 MHz clock input used to qualify the timing of the serial data being written or read
26	+5V	The +5 volt supply
27	SEEKPH3	Disk drive register write strobe LSTRB
28	SEEKPH1	Disk drive register select line CA1

The Apple IIc Plus multidrive interface glue (MIG)

The MIG is actually two ICs: a custom integrated circuit (the *MIG chip*) and a 2 KB static RAM. The static RAM is used as a RAM buffer to support the high data rate of the 3.5-inch disk drives. The MIG chip provides the interface (the "glue logic") that implements the RAM buffer for the internal disk drive and some interface logic for other disk drives. Figure 11-13 shows the pin assignments for the MIG chip, and Table 11-15 describes the pin functions.

■ **Figure 11-13** MIG chip pinouts



■ **Table 11-15** MIG chip signal descriptions

Pin	Signal	Description
1	IOSEL*	Tied to RESET* (pin 14)
2	A5	System address bit 5
3	A6	System address bit 6
4	A7	System address bit 7
5	A9	System address bit 9
6	A10	System address bit 10
7	A11	System address bit 11
8	BUSEN*	Output that indicates valid access to MIG address space
9	R/W	System read/write control line
10	PH1	System clock
11	EN2X*	Input that indicates an access to address $C0nx is occurring, where n is $9 through $E; an access to $C0nx addresses port n–8
12	VSS	Ground
13	ROMEN1*	Input that indicates an access to lower ROM bank space $C100 through $DFFF is occurring
14	RESET*	System reset or ROM address 14
15	IWMRES*	*Latched output bit to reset IWM*
16	INTEN*	Latched output bit to enable internal drive
17	ENB2	Follows EN2X* or ROMEN1* if addressing mapped device space to select external drives
18	3.5 DRIVE*	Latched output bit to select 5.25-inch or 3.5-inch drives
19	HDSEL	Latched output bit used for 3.5-inch drives
20	ROMEN2*	ROM enable output line; follows ROMEN1*
21	RAMEN*	RAM enable output line to 2 KB RAM buffer
22	RAMA10	RAM buffer address bit 10
23	RAMA9	RAM buffer address bit 9
24	RAMA8	RAM buffer address bit 8
25	VDD	+5 VDC
26	RAMA7	RAM buffer address bit 7
27	RAMA6	RAM buffer address bit 6
28	RAMA5	RAM buffer address bit 5

The Apple IIc Plus cache glue gate array (CGGA)

Most computer programs run the same routines repeatedly; therefore, a data cache, which allows fast access to the most recently used data, can often significantly enhance the speed of a system. In the Apple IIc Plus computer, a data cache is combined with a 4 MHz 65C02 microprocessor, so that frequently used data and code can be processed at 4 MHz instead of the 1.023 MHz processor speed of earlier Apple IIc computers.

The Apple IIc Plus accelerator circuit includes the Cache Glue Gate Array (CGGA), a 16 MHz crystal oscillator, and two 8 KB static RAM chips. As shown in the Apple IIc Plus block diagram, Figure 11-2, the CGGA interfaces the cache RAM and 65C02 to the rest of the system. The CGGA provides a 4 MHz clock to the CPU. When the accelerator is enabled and the CPU is processing data from the cache RAM, the system runs at its fastest speed. When the CPU has to access memory from system RAM, or when the accelerator is disabled, the CGGA synchronizes the CPU to the rest of the system and the Apple IIc Plus runs at the same speed as other Apple IIc computers.

The CGGA is programmed to keep track of which parts of the address space can be cached and which cannot. For example, when the Apple IIc Plus is switched on or reset, ROM code for ports 1, 2, 5, and 6, and the code for the speaker and game paddles cannot be cached; this code is restricted to run at 1.023 MHz. The code for ports 3, 4, and 7 can be cached, and so can run at up to 4 MHz.

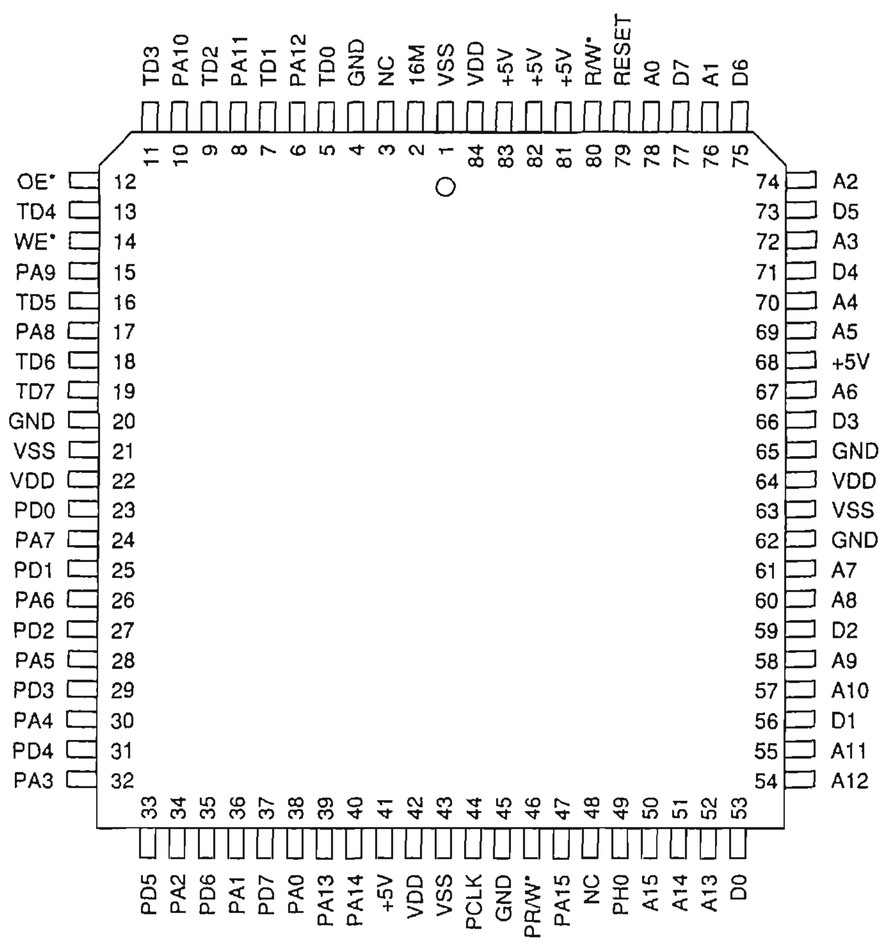
The RAM cache consists of two 8 KB static RAM chips: one RAM for data and one for tags. For each byte of data in the data RAM there is a corresponding tag byte in the tag RAM. Because there is 128 KB of system memory in the Apple IIc Plus and only 8 KB of memory in the data cache, the tag RAM is used to indicate from which part of memory each byte of data came. The CGGA keeps track of which areas of bank-switched memory are switched in, and determines from the tags whether the data in the data RAM corresponds to the address on the processor address bus. If the tag for a byte of data indicates that the byte is the one requested by the 65C02, the data is said to be *valid*.

When the 65C02 initiates a data read operation by raising its R/W line, the CGGA checks the tag RAM to determine if the data in the RAM cache is valid. If the data is valid and the accelerator is enabled, the CGGA provides a 4 MHz clock to the 65C02 and puts the data from the RAM cache on the processor data bus. If, however, the CGGA determines that the data in the RAM cache is not valid, the CGGA provides a 1.023 MHz clock to the 65C02, places the address of the data on the system address bus, and reads the data from system RAM. The CGGA then returns the data from memory to the 65C02, updates the data cache with the same data, and updates the tag cache.

When the 65C02 writes data to memory, the CGGA updates the data and tag caches as well as system memory. If writes are separated by 12 or more machine cycles, the CGGA acts as an interface to allow asynchronous operation of the the processor bus and the system bus. In this case, the 65C02 can continue running at 4 MHz while the CGGA writes the data to system RAM at 1.023 MHz. Each time firmware or software accesses port 1, 2, 5, or 6 by addressing $C0nx (where n is the port number + 8), the Apple IIc Plus switches to 1.023 MHz operation for approximately 50 ms. When the speaker or game paddle is accessed, the Apple IIc Plus switches to 1.023 MHz operation for approximately 5 ms. The Apple IIc Plus does not switch back to 4 MHz operation until approximately 50 ms after the *last* access to $C0nx (or 5 ms after the last access to the speaker or game paddle).

Figure 11-14 shows the pinout assignments for the Apple IIc Plus CGGA chip. Table 11-16 describes the pin functions for the CGGA.

■ **Figure 11-14** Apple IIc Plus accelerator chip pinouts



■ **Table 11-16** Apple IIc Plus accelerator chip signal descriptions

Pin	Signal	In/Out	Description
1	VSS	Input	Power and signal ground reference
2	16M	Input	Master clock for CGGA
3	NC	Output	Not connected
4	CI	Input	Ground
5	TD0	I/O	Tag RAM data bit 0
6	PA12	Input	Processor address bit 12
7	TD1	I/O	Tag RAM data bit 1
8	PA11	Input	Processor address bit 11
9	TD2	I/O	Tag RAM data bit 2
10	PA10	Input	Processor address bit 10
11	TD3	I/O	Tag RAM data bit 3
12	OE*	Output	Tag and data RAM output enable
13	TD4	I/O	Tag RAM data bit 4
14	WE*	Output	Tag and data RAM write enable
15	PA9	Input	Processor address bit 9
16	TD5	I/O	Tag RAM data bit 5
17	PA8	Input	Processor address bit 8
18	TD6	I/O	Tag RAM data bit 6
19	TD7	I/O	Tag RAM data bit 7
20	DD	Input	Ground
21	VSS	Input	Power and signal ground reference
22	VDD	Input	+5V
23	PD0	I/O	Processor bus data bit 0
24	PA7	Input	Processor address bit 7
25	PD1	I/O	Processor bus data bit 1
26	PA6	Input	Processor address bit 6
27	PD2	I/O	Processor bus data bit 2
28	PA5	Input	Processor address bit 5
29	PD3	I/O	Processor bus data bit 3
30	PA4	Input	Processor address bit 4
31	PD4	I/O	Processor bus data bit 4
32	PA3	Input	Processor address bit 3
33	PD5	I/O	Processor bus data bit 5
34	PA2	Input	Processor address bit 2
35	PD6	I/O	Processor bus data bit 6
36	PA1	Input	Processor address bit 1
37	PD7	I/O	Processor bus data bit 7

■ **Table 11-16** Apple IIc Plus accelerator chip signal descriptions (continued)

Pin	Signal	In/Out	Description
38	PA0	Input	Processor address bit 0
39	PA13	Input	Processor address bit 13
40	PA14	Input	Processor address bit 14
41	S1	Input	Tied to +5V
42	VDD	Input	+5V
43	VSS	Input	Power and signal ground reference
44	PCLK	Output	Processor clock output (0–4 MHz)
45	BS	Input	Ground
46	PR/W*	Input	Processor read/write signal
47	PA15	Input	Processor address bit 15
48	NC	Output	Not connected
49	PH0	Input	System clock, phase 0
50	A15	Output	System address bit 15
51	A14	Output	System address bit 14
52	A13	Output	System address bit 13
53	D0	I/O	System data bit 0
54	A12	Output	System address bit 12
55	A11	Output	System address bit 11
56	D1	I/O	System data bit 1
57	A10	Output	System address bit 10
58	A9	Output	System address bit 9
59	D2	I/O	System data bit 2
60	A8	Output	System address bit 8
61	A7	Output	System address bit 7
62	T1	Input	Ground
63	VSS	Input	Power and signal ground reference
64	VDD	Input	+5V
65	C2	Input	Ground
66	D3	I/O	System data bit 3
67	A6	Output	System address bit 6
68	SPKR	Input	Tied to +5V
69	A5	Output	System address bit 5
70	A4	Output	System address bit 4
71	D4	I/O	System data bit 4
72	A3	Output	System address bit 3
73	D5	I/O	System data bit 5
74	A2	Output	System address bit 2

■ **Table 11-16** Apple IIc Plus accelerator chip signal descriptions (continued)

Pin	Signal	In/Out	Description
75	D6	I/O	System data bit 6
76	A1	Output	System address bit 1
77	D7	I/O	System data bit 7
78	A0	Output	System address bit 0
79	RESET*	Input	System reset input
80	R/W*	Output	System R/W signal
81	S3	Input	Tied to +5V
82	S4	Input	Tied to +5V
83	S7	Input	Tied to +5V
84	VDD	Input	+5V

Enabling and disabling the accelerator

When the user presses the Reset key, the system reset handler first checks the state of the Esc key. If the Esc key is down, the system reset handler sets the Apple IIc Plus to run at 1.023 MHz. If the Reset key was pressed, the reset handler then checks the Control and Command (Open Apple) keys. If the Control key is not down, the reset handler restores the system to its state before the Reset key was pressed. If the Control key is down, the system reset handler performs a warm reset; if the Control and Command keys are both down, the system performs a cold reset.

The accelerator is automatically enabled (that is, the system runs at 4 MHz) whenever the Apple IIc Plus is started up, and whenever the computer is reset unless the Esc key is held down before and during the reset operation.

◆ *Note:* If your program uses the Esc key as a menu item, it should clear the keyboard strobe before looking for the first menu choice. Clearing the keyboard strobe prevents your program from interpreting the Esc key as a menu choice in the event that the user pressed Esc on reset in order to set the machine to 1.023 MHz. This problem is handled automatically by ProDOS 1.5.

Your program can temporarily disable the accelerator to ensure that a section of code runs at 1.023 MHz. If your program reads address $C0D0, the Apple IIc Plus switches to 1.023 MHz operation for 50 ms. For a 5 ms period of 1.023 MHz operation, you can access $C070 from any program that does not use the game paddles. An access to $C070 causes game paddles to reset.

The ROM Wait routine

The Wait routine at $FCA8 in the ROM runs at 1.023 MHz regardless of the speed to which the accelerator is set. The machine continues to run at 1.023 MHz for up to 50 ms after exiting the Wait routine. The Wait routine provides a minimum delay, not an absolute time. For more information on the Wait routine, see Appendix F.

Memory addressing

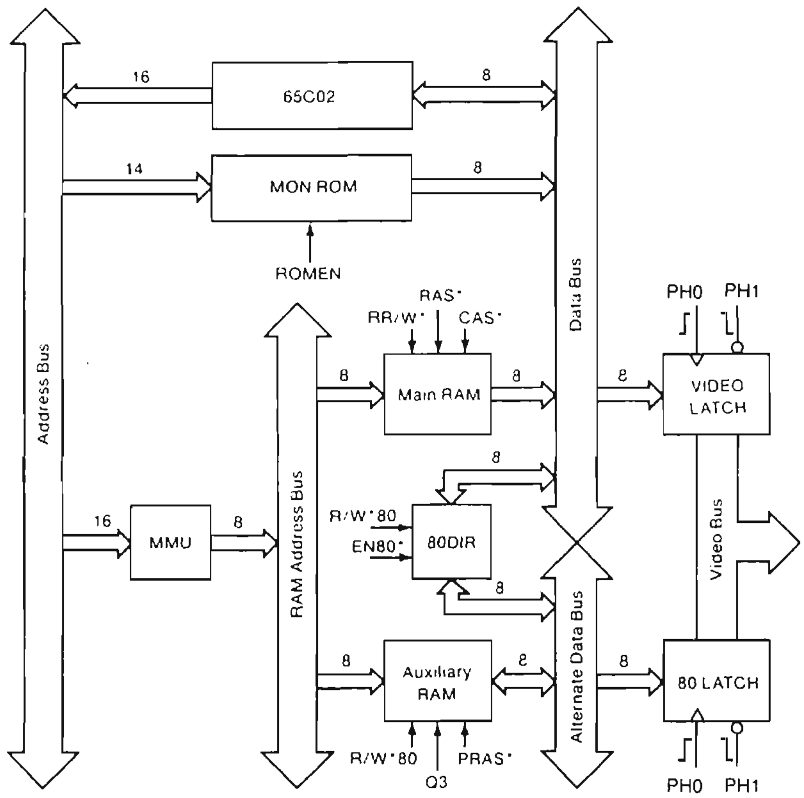
Each Apple IIc family member has 128 KB of system RAM and 32 KB of system ROM (16 KB in the original Apple IIc). The Apple IIc Plus computer has an additional 2 KB RAM buffer for the 3.5-inch disk drives and two 8 KB RAM chips for the accelerator circuit; however these additional RAMs are not available for use by programs. The two 8 KB RAM chips are not directly addressable, and the 2 KB RAM buffer is dedicated to the disk drive interface.

The 65C02 processor can directly address only 64 KB of memory; therefore each address is used for more than one function. The following sections describe the memory devices used in the Apple IIc family and the way they are addressed. Hardware devices also use portions of the memory address space; refer to Chapter 2 for information.

Figure 11-15 illustrates the overall memory bus organization and memory selection signals in Apple IIc family computers.

◆ *Note:* Some Apple IIc computers have ROM chips with 27xx designations, while others have 23xx. They are functionally equivalent.

■ **Figure 11-15** Memory bus organization

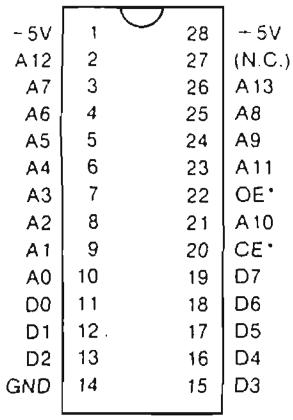


ROM addressing

The operating firmware for the Apple IIc family computers is permanently stored in a type 23256 (32 KB) ROM IC—called the *Monitor ROM*—as shown in Figure 11-16 and Table 11-17. The operating firmware is all of the firmware routines that "run" the Apple IIc family computers, like Applesoft, the Monitor, and the video firmware.

△ Original IIc The original Apple IIc uses a smaller 23128 (16 KB) ROM IC for the Monitor ROM. △

■ **Figure 11-16** 23256 ROM pinouts



■ **Table 11-17** 23256 ROM signal descriptions

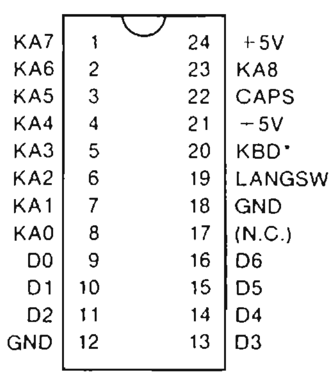
Pin	**Signal**	**Description**
1	+5V	+5 volts
2	A12	System address line 12
3	A7	System address line 7
4	A6	System address line 6
5	A5	System address line 5
6	A4	System address line 4
7	A3	System address line 3
8	A2	System address line 2
9	A1	System address line 1
10	A0	System address line 0
11	D0	System data line 0
12	D1	System data line 1
13	D2	System data line 2
14	GND	Ground
15	D3	System data line 3
16	D4	System data line 4
17	D5	System data line 5
18	D6	System data line 6
19	D7	System data line 7
20	CE*	Chip enable
21	A10	System address line 10
22	OE*	Output enable

■ **Table 11-17** 23256 ROM signal descriptions (continued)

Pin	Signal	Description
23	A11	System address line 11
24	A9	System address line 9
25	A8	System address line 8
26	A13	System address line 13
27	RA14	ROM address line 14 (from IOU; this pin is not connected on 23128 ROM)
28	+5V	+5 volts

The other ROMs in the Apple IIc family computers are a type 2316 ROM (see Figure 11-17 and Table 11-18) used for the keyboard character decoder, and a type 2364 ROM (see Figure 11-18 and Table 11-19) used for character sets for the video display. The 2316 ROM is called the *keyboard ROM* and the 2364 ROM is called the *character generator ROM*. The 2364 ROM is rather large because it includes a section of straight-through bit mapping for the graphics modes. This way, graphics display video can pass through the same circuits as text without additional switching circuitry.

■ **Figure 11-17** 2316 ROM pinouts



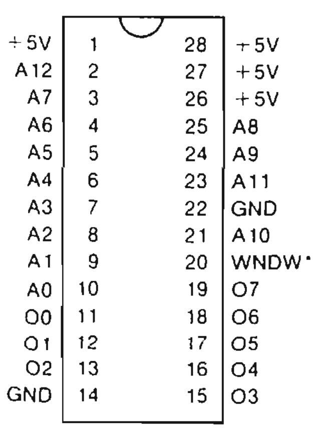
■ **Table 11-18** 2316 ROM signal descriptions

Pin	Signal	Description
1	KA7	Keyboard address line 7
2	KA6	Keyboard address line 6
3	KA5	Keyboard address line 5
4	KA4	Keyboard address line 4

■ **Table 11-18** 2316 ROM signal descriptions (continued)

Pin	Signal	Description
5	KA3	Keyboard address line 3
6	KA2	Keyboard address line 2
7	KA1	Keyboard address line 1
8	KA0	Keyboard address line 0
9	D0	System data line 0
10	D1	System data line 1
11	D2	System data line 2
12	GND	Ground
13	D3	System data line 3
14	D4	System data line 4
15	D5	System data line 5
16	D6	System data line 6
17	N.C.	Not connected
18	GND	Ground
19	LANGSW	Language switch
20	KBD'	Enables keyboard data lines
21	+5V	+5 volts
22	CAPS	Caps Lock
23	KA8	Keyboard address line
24	+5V	+5 volts

■ **Figure 11-18** 2364 ROM pinouts



■ **Table 11-19** 2364 ROM signal descriptions

Pin	Signal	Description
1	+5V	+5 volts
2	LANGSW	Language switch
3	VIDD4	Video data line 4
4	VIDD3	Video data line 3
5	VIDD2	Video data line 2
6	VIDD1	Video data line 1
7	VIDD0	Video data line 0
8	VC	VC bit from IOU
9	SEGB	SEGB bit from IOU
10	SEGA	SEGA bit from IOU
11	O0	Video output line 0
12	O1	Video output line 1
13	O2	Video output line 2
14	GND	Ground
15	O3	Video output line 3
16	O4	Video output line 4
17	O5	Video output line 5
18	O6	Video output line 6
19	O7	Video output line 7
20	WNDW*	Display blanking signal
21	RA10*	Video display control bit
22	GND	Ground
23	GR	Graphics mode enable
24	RAS*	Row address strobe
25	VIDD5	Video data line 5
26-28	+5V	+5 volts

RAM addressing

The system RAM in the Apple IIc family computers is used both for program and data storage and for the video display. The areas in RAM that are used for the display are accessed both by the 65C02 microprocessor and by the video display circuits. In some computers, this dual access results in addressing conflicts (cycle stealing) that can cause temporary dropouts in the video display. This problem does not occur in the Apple IIc family computers, due to the way the microprocessor and the video circuits share the memory.

The memory circuits in the Apple IIc family computers take advantage of the two-phase system clock to interleave the microprocessor memory accesses and the display memory accesses so that they never interfere with each other. The microprocessor reads or writes to RAM only during PH0, and the display circuits read data only during PH1.

Dynamic RAM refreshment

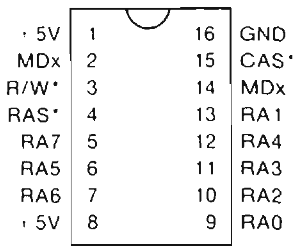
The image on a video display is not permanent; it fades rapidly and must be refreshed periodically. To refresh the video display, each Apple IIc family computer reads the data in the active display page and sends them to the display. To prevent visible flicker in the display, and to conform to standard practice for broadcast video, the Apple IIc family computer refreshes the display 60 times per second.

The dynamic RAM devices used in the Apple IIc family computers also need a kind of refreshment, because the data are stored in the form of electric charges that diminish with time and must be replenished. The Apple IIc family computers are designed so that refreshing the display also refreshes the dynamic RAMs. The next few paragraphs explain how this is done.

The job of refreshing the dynamic RAM devices is minimized by the structure of the devices themselves. The individual data cells in each RAM device are arranged in a rectangular array of rows and columns. When the device is addressed, the part of the address that specifies a row is presented first, followed by the address of the column. Splitting information into parts that follow each other in time is called ***multiplexing***. Because only half of the address is needed at one time, multiplexing the address reduces the number of pins needed for connecting the RAMs. The pinouts of the 64 KB RAM chip are shown in Figures 11-19 and Table 11-20. The pinouts of the 256 Kbit RAM chip are shown in Figure 11-20 and Table 11-21.

△ Apple IIc Plus Memory expansion: The memory expansion Apple IIc and the Apple IIc Plus replace the sixteen 64 KBit RAM ICs with four 256 Kbit RAM ICs. These RAM ICs are functionally identical to the ICs they replace. △

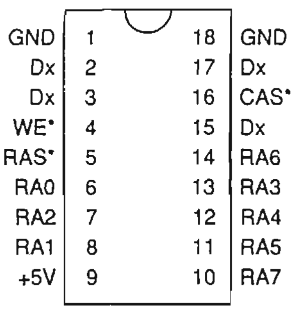
■ **Figure 11-19** 64 Kbit RAM pinouts



■ **Table 11-20** 64 Kbit RAM signal descriptions

Pin	Signal	Description
1	+5V	+5 volts
2	Dx	Data line
3	R/W*	Read/write
4	RAS*	Row address strobe
5	RA7	RAM address line 7
6	RA5	RAM address line 5
7	RA6	RAM address line 6
8	+5V	+5 volts
9	RA0	RAM address line 0
10	RA2	RAM address line 2
11	RA3	RAM address line 3
12	RA4	RAM address line 4
13	RA1	RAM address line 1
14	Dx	Data line
15	CAS*	Column address strobe
16	GND	Ground

■ **Figure 11-20** Pinouts of 256 Kbit RAM chip



■ **Table 11-21** 256 Kbit RAM signal descriptions

Pin	Signal	Description
1	GND	Ground
2	Dx	Data line
3	Dx	Data line
4	WE•	Write enable
5	RAS•	Row address strobe
6	RA0	RAM address line 0
7	RA2	RAM address line 2
8	RA1	RAM address line 1
9	+5V	+5 Volts
10	RA7	RAM address line 7
11	RA5	RAM address line 5
12	RA4	RAM address line 4
13	RA3	RAM address line 3
14	RA6	RAM address line 6
15	Dx	Data line
16	CAS•	Column address strobe
17	Dx	Data line
18	GND	Ground

Different manufacturers' 64 Kbit RAMs have cell arrays of either 128 rows by 512 columns or 256 rows by 256 columns. Only the row portion of the address is used in refreshing the RAMs.

Now consider how the display is refreshed. As described later in this chapter, the display circuitry generates a sequence of 8,192 memory addresses in Hi-Res mode; in text and Lo-Res modes, this sequence is the 1,024 display-page addresses repeated 8 times. The display address cycles through this sequence 60 times a second, or once every 17 milliseconds. The way the low-order address lines are assigned to the RAMs, the row address cycles through all 256 possible values once every 2 milliseconds (see Table 11-22). This more than satisfies the refresh requirements of the dynamic RAMs.

■ **Table 11-22** RAM address multiplexing

Mux'd address	Row address	Column address
RA0	A0	A9
RA1	A1	A6
RA2	A2	A10
RA3	A3	A11
RA4	A4	A12
RA5	A5	A13
RA6	A7	A14
RA7	A8	A15

Dynamic RAM timing

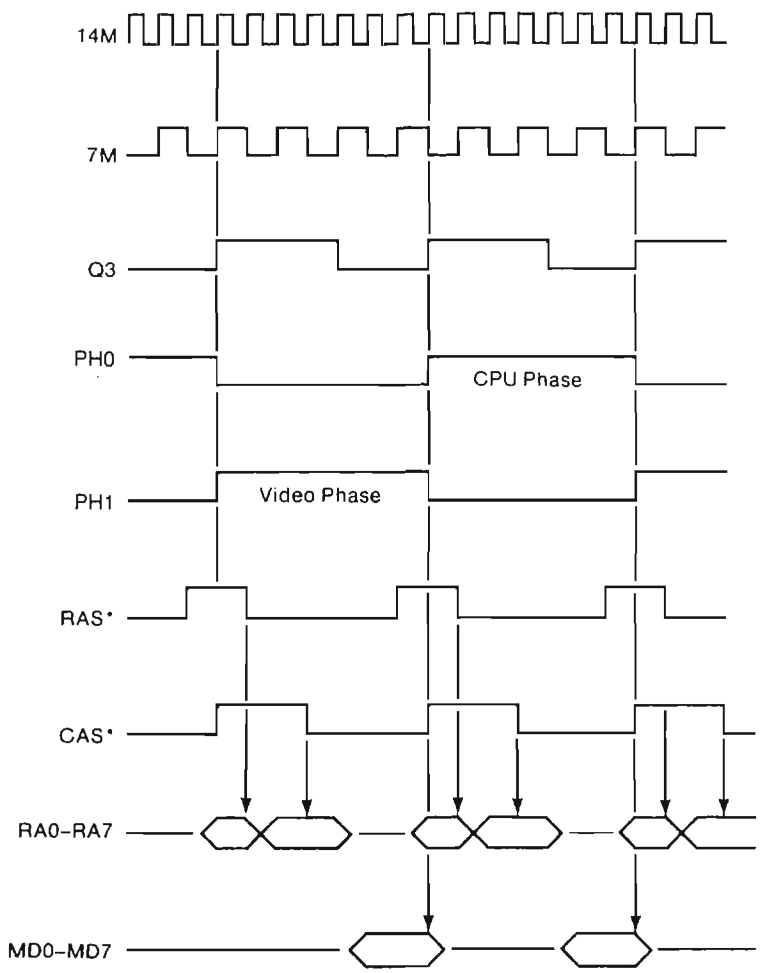
The microprocessor clock of the Apple IIc runs at a speed of 1.023 MHz, but the interleaving of CPU and display cycles means that the RAM is being accessed at a 2 MHz rate, or a cycle time of just under 500 nanoseconds. Data for the CPU are strobed by the falling edge of PH0, and display data are strobed by the falling edge of PH1, as shown in Figure 11-21.

The RAM timing looks complicated because the RAM address is multiplexed, as described previously. The MMU takes care of multiplexing the address for the CPU cycle, and the IOU performs the same function for the display cycle. The multiplexed address is sent to the RAM ICs over the lines RA0–RA7 (Table 11-23). Along with the other timing signals, the TMG generates two signals that control the RAM addressing: row-address strobe (RAS) and column-address strobe (CAS).

■ **Table 11-23** RAM timing signals

Signal	Description
PH0	System clock, phase 0 (CPU phase)
PH1	System clock, phase 1 (video phase)
RAS	Row-address strobe
CAS	Column-address strobe
Q3	Alternate RAM/column-address strobe
RA0–RA7	Multiplexed address bus
MD0–MD7	Internal data bus

■ **Figure 11-21** RAM timing signals

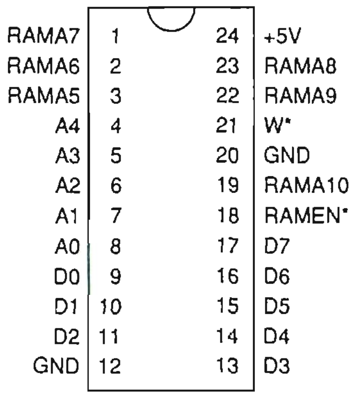


The Apple IIc Plus MIG RAM

The MIG circuit in the Apple IIc Plus computer uses a 16 KBit static RAM chip to provide a RAM buffer for the 3.5-inch disk drives. This RAM, which does not have to be refreshed, is dedicated to the disk drive interface and should not be used by programs. The pinouts of the MIG RAM chip are shown in Figure 11-22 and Table 11-24.

▲ **Warning** Do not attempt to use the MIG RAM for program data; doing so may corrupt disk drive data. ▲

■ **Figure 11-22** MIG RAM pinouts



■ **Table 11-24** MIG RAM signal descriptions

Pin	Signal	Description
1	RAMA7	RAM address line 7
2	RAMA6	RAM address kube 6
3	RAMA5	RAM address line 5
4	A4	System address line 4
5	A3	System address line 3
6	A2	System address line 2
7	A1	System address line 1
8	A0	System address line 0
9	D0	Data line 0
10	D1	Data line 1
11	D2	Data line 2
12	GND	Ground
13	D3	Data line 3
14	D4	Data line 4
15	D5	Data line 5
16	D6	Data line 6
17	D7	Data line 7
18	RAMEN*	RAM enable

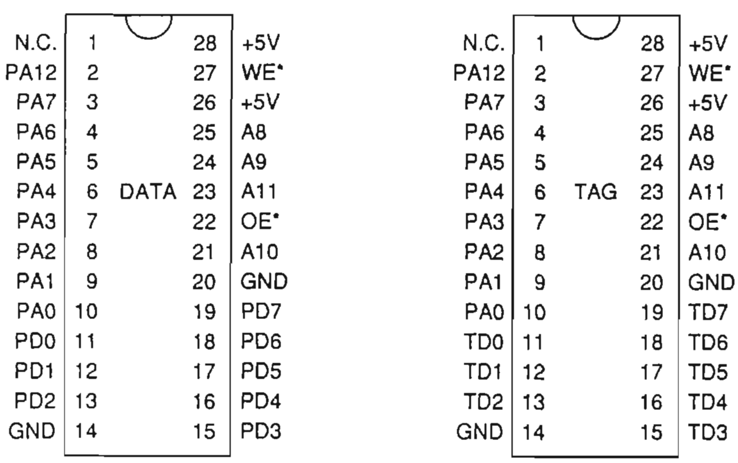
■ **Table 11-24** MIG RAM signal descriptions (continued)

Pin	Signal	Description
19	RAMA10	RAM address line 10
20	GND	Ground
21	W*	Write
22	RAMA9	RAM address line 9
23	RAMA8	RAM address line 8
24	+5V	+5 volts

The Apple IIc Plus accelerator cache RAM

The Apple IIc Plus computer uses two 8 KB static RAM chips to provide a data cache for the accelerator circuit. This RAM is used for program instructions, operands, and data. The cache RAM, which does not have to be refreshed, cannot be directly addressed by programs. The pinouts of the cache RAM chips are shown in Figure 11-23. For the uses of these signals, see Table 11-25 and 11-26.

■ **Figure 11-23** Accelerator cache RAM pinouts



■ **Table 11-25** Cache data RAM signal descriptions

Pin	Signal	Description
1	N.C.	Not connected
2	PA12	Processor address line 12
3	PA7	Processor address line 7
4	PA6	Processor address line 6
5	PA5	Processor address line 5
6	PA4	Processor address line 4
7	PA3	Processor address line 3
8	PA2	Processor address line 2
9	PA1	Processor address line 1
10	PA0	Processor address line 0
11	PD0	Processor data line 0
12	PD1	Processor data line 1
13	PD2	Processor data line 2
14	GND	Ground
15	PD3	Processor data line 3
16	PD4	Processor data line 4
17	PD5	Processor data line 5
18	PD6	Processor data line 6
19	PD7	Processor data line 7
20	GND	Ground
21	A10	System address line 10
22	OE*	Output enable
23	A11	System address line 11
24	A9	System address line 9
25	A8	System address line 8
26	+5V	+5 volts
27	WE*	Write enable
28	+5V	+5 volts

■ **Table 11-26** Cache tag RAM signal descriptions

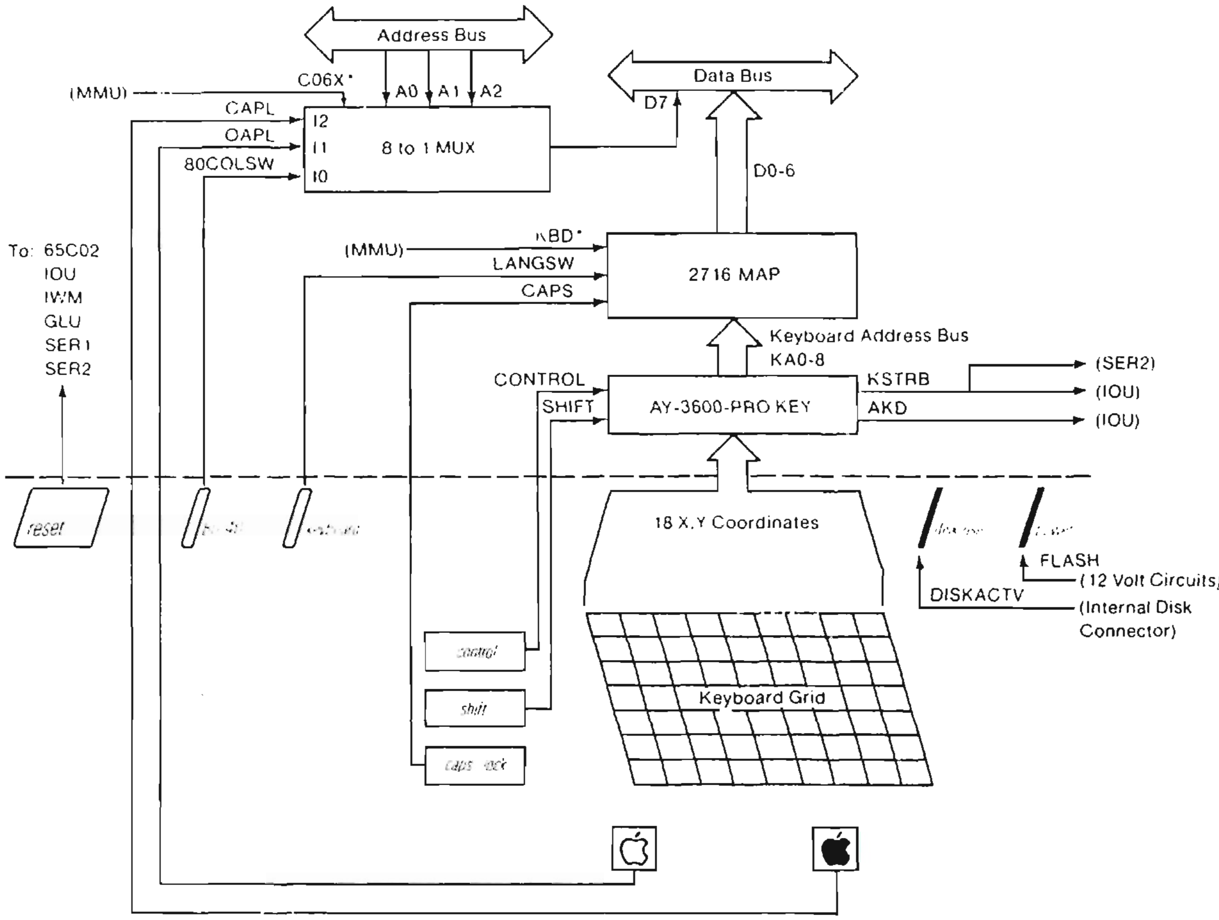
Pin	Signal	Description
1	N.C.	Not connected
2	PA12	Processor address line 12
3	PA7	Processor address line 7
4	PA6	Processor address line 6
5	PA5	Processor address line 5
6	PA4	Processor address line 4
7	PA3	Processor address line 3
8	PA2	Processor address line 2
9	PA1	Processor address line 1
10	PA0	Processor address line 0
11	TD0	Tag data line 0
12	TD1	Tag data line 1
13	TD2	Tag data line 2
14	GND	Ground
15	TD3	Tag data line 3
16	TD4	Tag data line 4
17	TD5	Tag data line 5
18	TD6	Tag data line 6
19	TD7	Tag data line 7
20	GND	Ground
21	A10	System address line 10
22	OE*	Output enable
23	A11	System address line 11
24	A9	System address line 9
25	A8	System address line 8
26	+5V	+5 volts
27	WE*	Write enable
28	+5V	+5 volts

The keyboard

The keyboard is a matrix of key switches connected to an AY-3600-type keyboard decoder via a ribbon cable and a 34-pin connector. Figure 11-24 is a block diagram of the keyboard circuit. The AY-3600 scans the array of keys over and over to detect any keys pressed. The scanning rate and debounce time are set by an external resistor-capacitor network.

The Apple IIc Plus keyboard has a slightly different key arrangement than that of the Apple IIc. The Apple IIc Plus has no 40/80 column switch, and the volume control is located on the keyboard rather than on the side of the machine. The Option key on the Apple IIc Plus keyboard is the same as the Solid Apple key on older Apple IIc keyboards. Functionally, the Apple IIc and Apple IIc Plus keyboards are identical.

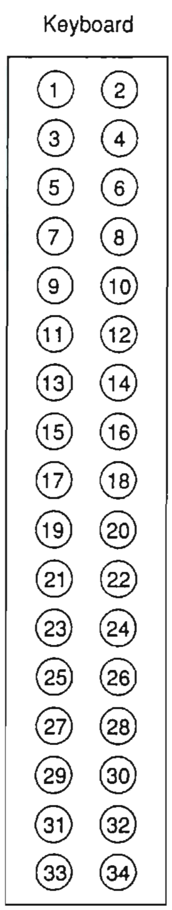
■ **Figure 11-24** Keyboard circuit block diagram



The AY-3600 outputs include five bits of key code plus separate lines for Control, Shift, any-key-down (AKD), and keyboard strobe (KSTRB). The AKD and KSTRB lines are connected to the IOU. The key-code line and Control and Shift are inputs to a separate 2316 ROM. (Some Apple IIc computers have a 2716 EPROM chip, which is functionally equivalent to the 2316 ROM chip.) The ROM translates the key codes to character codes, which are enabled onto the data bus by the KBD* signal. The KBD* signal is enabled by the MMU whenever a program reads location $C000.

Figure 11-25 shows the keyboard connector. Table 11-27 shows the signal assignments for the keyboard connector in the original, UniDisk 3.5, and memory expansion versions of the Apple IIc computer. Table 11-28 shows the signal assignments for the keyboard connector in the Apple IIc Plus computer.

■ **Figure 11-25** Keyboard connector



■ **Table 11-27** Apple IIc keyboard connector signals

Pin	Signal	Description
1	Y7	Y coordinate bit 7
2	Y9	Y coordinate bit 9
3	X4	X coordinate bit 4
4	Y8	Y coordinate bit 8
5	X5	X coordinate bit 5
6	Y5	Y coordinate bit 5
7	X6	X coordinate bit 6
8	Y3	Y coordinate bit 3
9	X7	X coordinate bit 7
10	Y6	Y coordinate bit 6
11	DISKACTY	Disk activity light
12	X0	X coordinate bit 0
14	Y4	Y coordinate bit 4
15	LANGSW	Keyboard (Dvorak) switch
16	CAPL	Option (Solid Apple) key†
18	X2	X coordinate bit 2
19	80COLSW	40/80 column switch
20	X3	X coordinate bit 3
21	GND	Ground
22	Y2	Y coordinate bit 2
23	+5V	+5 volts
24	Y1	Y coordinate bit 1
25	+5V	+5 volts
26	OAPL	Command (Open Apple) key
27	GND	Ground
28	CAPS	Caps Lock key
29	RESET*	Reset key
31	Y0	Y coordinate bit 0
32	CONTROL	Control key
33	X1	X coordinate bit 1
34	SHIFT	Shift key

† CAPL is an acronym for "Closed Apple," an alternate name for the Solid Apple key.

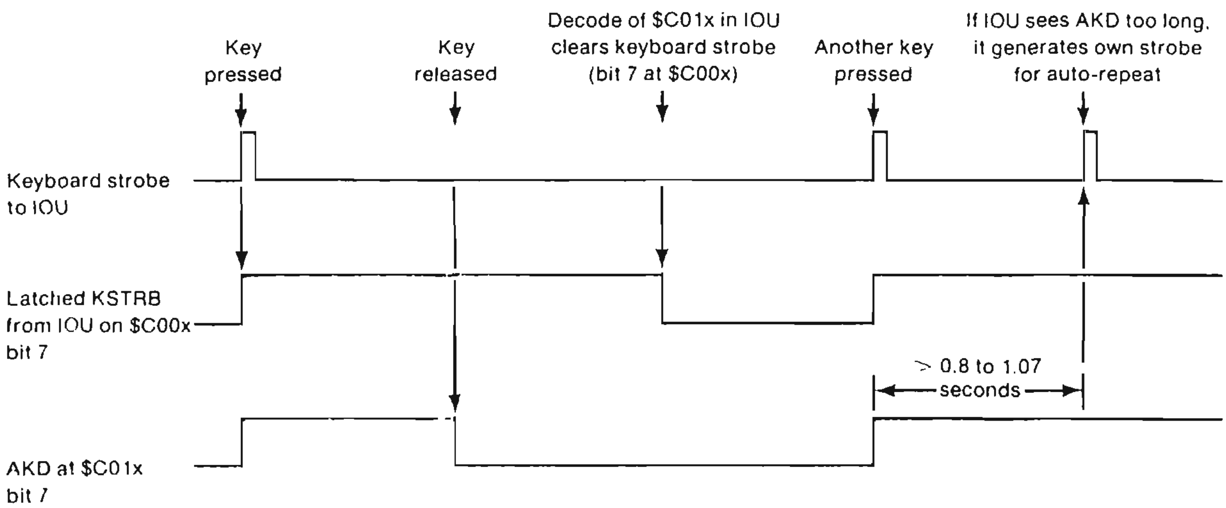
■ **Table 11-28** Apple IIc Plus keyboard connector signals

Pin	Signal	Description
1	Y7	Y coordinate bit 7
2	Y9	Y coordinate bit 9
3	X4	X coordinate bit 4
4	Y8	Y coordinate bit 8
5	X5	X coordinate bit 5
6	Y5	Y coordinate bit 5
7	X6	X coordinate bit 6
8	Y3	Y coordinate bit 3
9	X7	X coordinate bit 7
10	Y6	Y coordinate bit 6
11	DISKACTY	Disk activity light
12	X0	X coordinate bit 0
14	Y4	Y coordinate bit 4
15	VOLUME	Volume control
16	CAPL	Option key†
18	X2	X coordinate bit 2
19	LANGSW	Keyboard (Dvorak) switch
20	X3	X coordinate bit 3
21	GND	Ground
22	Y2	Y coordinate bit 2
23	+5V	+5 VDC
24	Y1	Y coordinate bit 1
25	+5V	+5 VDC
26	OAPL	Command (Open Apple) key
27	GND	Ground
28	CAPS	Caps lock key
29	RESET*	Reset key
31	Y0	Y coordinate bit 0
32	CONTROL	Control key
33	X1	X coordinate bit 1
34	SHIFT	Shift key

† CAPL is an acronym for "Closed Apple," a reference to the label on this key on older Apple IIc computers.

Figure 11-26 illustrates the events that occur when a key is pressed, when the keypress is detected by a program, and when a key is pressed and held for more than about a second.

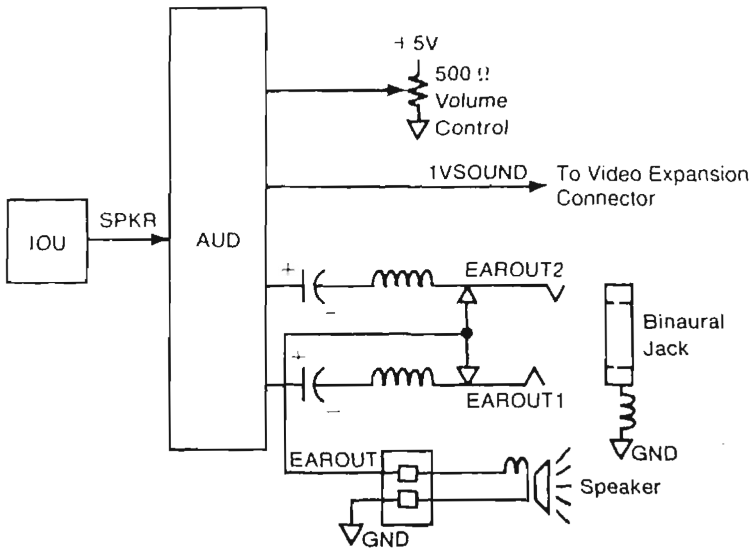
■ **Figure 11-26** Keyboard signals



The speaker

The built-in speaker is controlled by a single bit of output from the input/output unit (IOU). In the original, UniDisk 3.5, and memory expansion versions of the Apple IIc, this bit is input to the sound hybrid circuit (AUD) shown in Figure 11-27, which amplifies the pulse. In the Apple IIc Plus, the sound circuit is implemented as discrete components on the main logic board. The function of the sound circuit in the Apple IIc Plus is identical to that of the hybrid sound circuit in the Apple IIc, except that the external headphone jack is not supported in the Apple IIc Plus.

■ **Figure 11-27** Speaker circuit diagram



Volume control

There is a 500-ohm (1-kilohm in the Apple IIc Plus) variable resistor feeding anywhere from 0 to 5 volts to the speaker circuit to control the speaker volume.

The Apple IIc has a volume control knob on the side of the machine. The Apple IIc Plus has a sliding volume control on the keyboard.

Audio output jack

Next to the volume control, along the lower-left side of the Apple IIc case, there is a 3.5 mm audio output jack. Although speaker output is monaural, the jack accommodates stereo headphone plugs (as well as monaural), providing sound to both channels. Inserting a headphone plug into the jack disconnects the built-in speaker.

△ Apple IIc Plus The Apple IIc Plus has no audio output jack. △

The video display

The Apple IIc family computers produce a video signal that creates a display on a standard video monitor or, if you add an RF modulator, on a black-and-white or color television set. The video signal is a composite made up of the data that are being displayed plus the horizontal and vertical synchronization signals that the video monitor uses to arrange the lines of display data on the screen.

◆ *Note:* Apple IIc family computers manufactured for sale in the USA generate a video signal that is compatible with the standards set by the NTSC (National Television Standards Committee). Apple IIc family computers used in European countries require an external adapter to provide video that is compatible with the standard used there, which is called *PAL* (for *phase alternating lines*). References to the PAL standard are found in the bibliography at the end of this manual. This manual describes only the NTSC version of the video circuits.

The display portion of the video signal is a time-varying voltage generated from a stream of data bits, where a 1 corresponds to a voltage that generates a bright dot, and a 0 to a dark dot. The display bit stream is generated in bursts that correspond to the horizontal lines of dots on the video screen. The signal named WNDW' is low during these bursts.

During the time intervals between bursts of data, nothing is displayed on the screen. During these intervals, called the *blanking intervals*, the display is blank and the WNDW' signal is high. The synchronization signals, called *sync* for short, are produced by making the signal named SYNC' low during portions of the blanking intervals. The sync pulses are at a voltage equivalent to blacker-than-black video and don't show on the screen.

The video counters

The address and timing signals that control the generation of the video display are all derived from a chain of counters inside the IOU. Only a few of these counter signals are accessible from outside the IOU, but they are all important in understanding the operation of the display generation process, particularly the display memory addressing described in the next section.

The horizontal counter is made up of seven stages: H0, H1, H2, H3, H4, H5, and HPE'. The input to the horizontal counter is the 1 MHz signal that controls the reading of data being displayed. The complete cycle of the horizontal counter consists of 65 states. The six bits H0 through H5 count from 0 to 64, then start over at 0. Whenever this happens, HPE' forces another count with H0 through H5 held at 0, extending the total count to 65.

The IOU uses the 40 horizontal count values from 25 through 64 in generating the low-order part of the display data address. The IOU uses the count values from 0 to 24 to generate the horizontal blanking, the horizontal sync pulse, and the color-burst gate.

When the horizontal count gets to 65, it signals the end of a line by triggering the vertical counter. The vertical counter has nine stages: VA, VB, VC, V0, V1, V2, V3, V4, and V5. When the vertical count reaches 262, the IOU resets it and starts counting again from 0. Only the first 192 scanning lines are actually displayed; the IOU uses the vertical counts from 192 to 262 to generate the vertical blanking and sync pulse. Nothing is displayed during the vertical blanking interval. (The vertical line count is 262 rather than the standard 262.5 because, unlike normal television, the Apple IIc family's video display is not interlaced.)

Display memory addressing

As described in Chapter 6, data bytes are not stored in memory in the same sequence in which they appear on the display. You can get an idea of the way the display data are stored by using the Monitor to set the display to graphics mode, then storing data starting at the beginning of the display page at $0400 and watching the effect on the display. If you do this, you should use the graphics display instead of text to avoid confusion: the text display is also used for Monitor input and output.

If you want your program to display data by storing them directly into the display memory, you must first transform the display coordinates into the appropriate memory addresses, as shown in Chapter 2. The descriptions that follow will help you understand how this address transformation is done and why it is necessary.

The address transformation that folds three rows of 40 display bytes into 128 contiguous memory locations is the same for all display modes, so it is described first. The differences among the different display modes are described later in this chapter.

Display address mapping

Consider the simplest display on an Apple IIc family computer, the 40-column text mode. To address 40 columns requires 6 bits, and to address 24 rows requires another 5 bits, for a total of 11 address bits. Addressing the display this way would involve 2048 (2 to the 11th power) bytes of memory to display a mere 960 characters. The 80-column text mode would require 4096 bytes to display 1920 characters. The leftover chunks of memory that were not displayed could be used for storing other data, but not easily, because they would not be contiguous.

Instead of using the horizontal and vertical counts to address memory directly, the circuitry inside the IOU transforms them into the new address signals described below. The transformed display address must meet the following criteria:

- map the 960 bytes of 40-column text into only 1024 bytes
- scan the low-order address to refresh the dynamic RAMs
- continue to refresh the RAMs during video blanking

See the section "Dynamic RAM Refreshment" earlier in this chapter, for a discussion of RAM refreshing.

The transformation involves only horizontal counts H3, H4, and H5, and vertical counts V3 and V4. Vertical count bits VA, VB, and VC address the lines making up the characters, and are not involved in the address transformation. The remaining low-order count bits, H0, H1, H2, V0, V1, and V2 are used directly and are not involved in the transformation.

The IOU performs an addition that reduces the five significant count bits to four new signals S0, S1, S2, and S3, where *S* stands for *sum*. Figure 11-28 is a diagram showing the addition in binary form, with V3 appearing as the carry in and H5 appearing as its complement H5'. A constant value of 1 appears as the low-order bit of the addend. The carry bit generated with the sum is not used.

If this transformation seems terribly obscure, try it with actual values. For example, for the upper-left corner of the display, the vertical count is 0 and the horizontal count is 24: H0, H1, H2, and H5 are 0's, and H3 and H4 are 1's. The value of the sum is 0, so the memory location for the first character on the display is the first location in the display page, as you might expect.

■ **Figure 11-28** Display address transformation

			V3	Carry in
¬H5	V3	H4	H3	Augend
V4	¬H5	V4	1	Addend
S3	S2	S1	S0	Sum

Horizontal bits H0, H1, and H2 and sum bits S0, S1, and S2 make up the transformed horizontal address (A0 through A6 in Table 11-29). As the horizontal count increases from 24 to 63, the value of the sum (S3 S2 S1 S0) increases from 0 to 4 and the transformed address goes from 0 to 39, relative to the beginning of the display page.

■ **Table 11-29** Display memory addressing

Memory	Display address bit
A0	H0
A1	H1
A2	H2
A3	S0
A4	S1
A5	S2
A6	S3
A7	V0
A8	V1
A9	V2
A10	*
A11	*
A12	*
A13	*
A14	*
A15	GND

*For these address bits, see Table 11-30.

The low-order three bits of the vertical row counter are V0, V1, and V2. These bits control address bits A7, A8, and A9, as shown in Table 11-29, so that rows 0 through 7 start on 128-byte boundaries. When the vertical row counter reaches 8, V0, V1, and V2 are 0 again, and V3 changes to 1. If you do the addition in Figure 11-28 with H equal to 24 (the horizontal count for the first column displayed) and V equal to 8, the sum is 5 and the horizontal address is 40: the first character in row 8 is stored in the memory location 40 bytes from the beginning of the display page.

Table 11-29 shows how the signals from the video counters are assigned to the address lines. H0, H1, and H2 are horizontal-count bits, and V0, V1, and V2 are vertical-count bits. S0, S1, S2, and S3 are the folded address bits described earler. Table 11-30 shows memory address bits for the display modes.

■ **Table 11-30** Memory address bits for display modes

Address bit	Text and Lo-Res	Hi-Res and Double Hi-Res
A10	80Store AND NOT Page2	VA
A11	/80Store AND Page2†	VB
A12	0	VC
A13	0	80Store AND NOT Page2
A14	0	80Store AND Page2

Note: AND and NOT refer to the logical (Boolean) AND and inverse operations.

Figure 11-29 shows how groups of three 40-character rows are stored in blocks of 120 contiguous bytes starting on 128-byte address boundaries. This diagram is another way of describing the display mapping shown in Chapter 6. Notice that the three rows in each block of 120 bytes are not adjacent on the display.

■ **Figure 11-29** 40-column text display memory

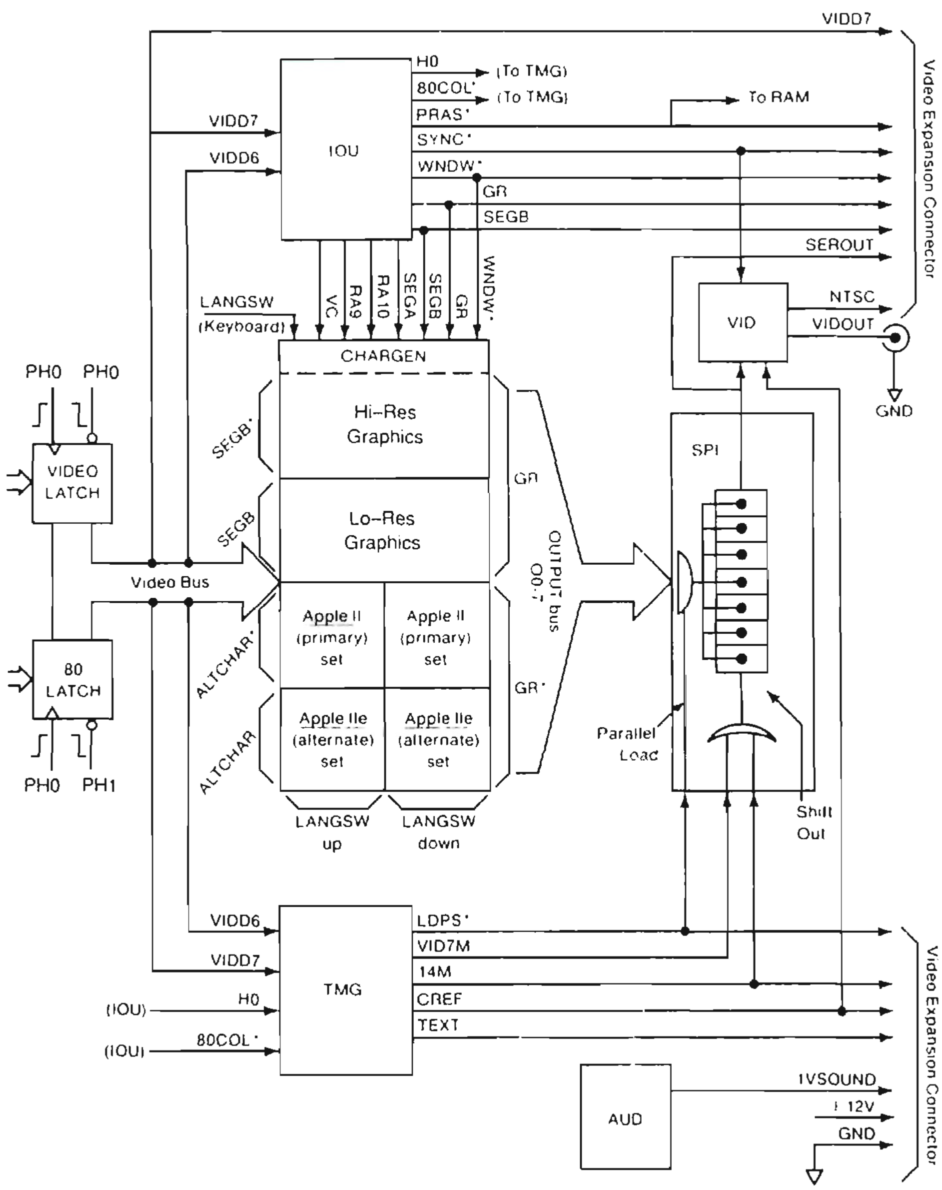
Memory locations marked with a double asterisk (**) are screen holes, described in Chapter 2.

	128 Bytes			
	40 Bytes	40 Bytes	40 Bytes	8 Bytes
$400	Row 0	Row 8	Row 16	**
$480	Row 1	Row 9	Row 17	**
$500	Row 2	Row 10	Row 18	**
$580	Row 3	Row 11	Row 19	**
$600	Row 4	Row 12	Row 20	**
$680	Row 5	Row 13	Row 21	**
$700	Row 6	Row 14	Row 22	**
$780	Row 7	Row 15	Row 23	**

Video display modes

The different display modes all use the address-mapping scheme described in the preceding section, but they use different-sized memory areas in different locations. This section describes the addressing schemes and the methods of generating the actual video signals for the different display modes. Figure 11-30 illustrates the video display circuits discussed in this section.

■ **Figure 11-30** Video display circuits



Text displays

The Text and Lo-Res graphics pages begin at memory locations $0400 and $0800. Table 11-30 shows how the display-mode signals control the address bits to produce these addresses. Address bits A10 and A11 are controlled by the settings of Page2 and 80Store, the display-page and 80-column-video soft switches. Address bits A12, A13, and A14 are set to 0. Note that 80Store active inhibits Page2: there is only one display page in 80-column mode.

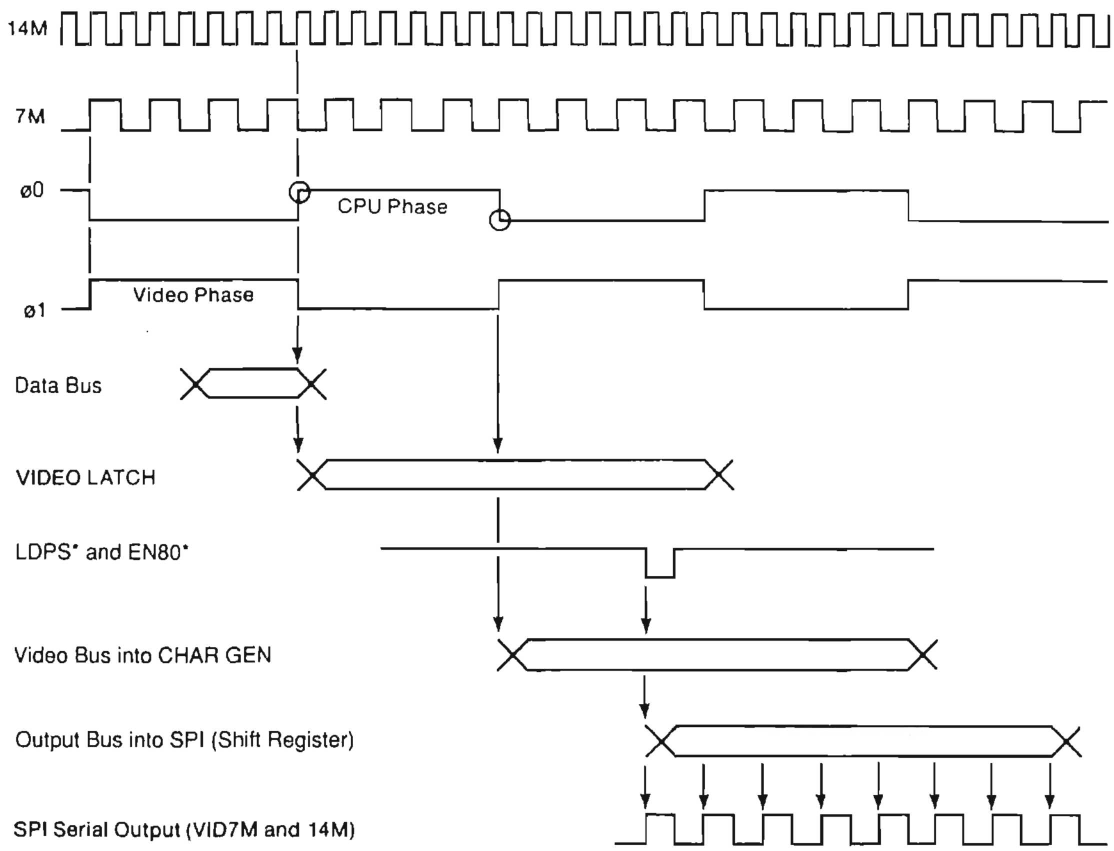
The low-order 6 bits of each data byte reach the character generator directly, via the video data bus VID0–VID5. The two high-order bits are modified by the IOU to select between the primary and alternate character sets and are sent to the character generator on lines RA9 and RA10.

The data for each row of characters are read eight times, once for each of the eight lines of dots making up the row of characters. The data bits are sent to the character generator along with VA, VB, and VC, the low-order bits from the vertical counter. For each character being displayed, the character generator puts out one of eight stored bit patterns selected by the 3-bit number made up of VA, VB, and VC.

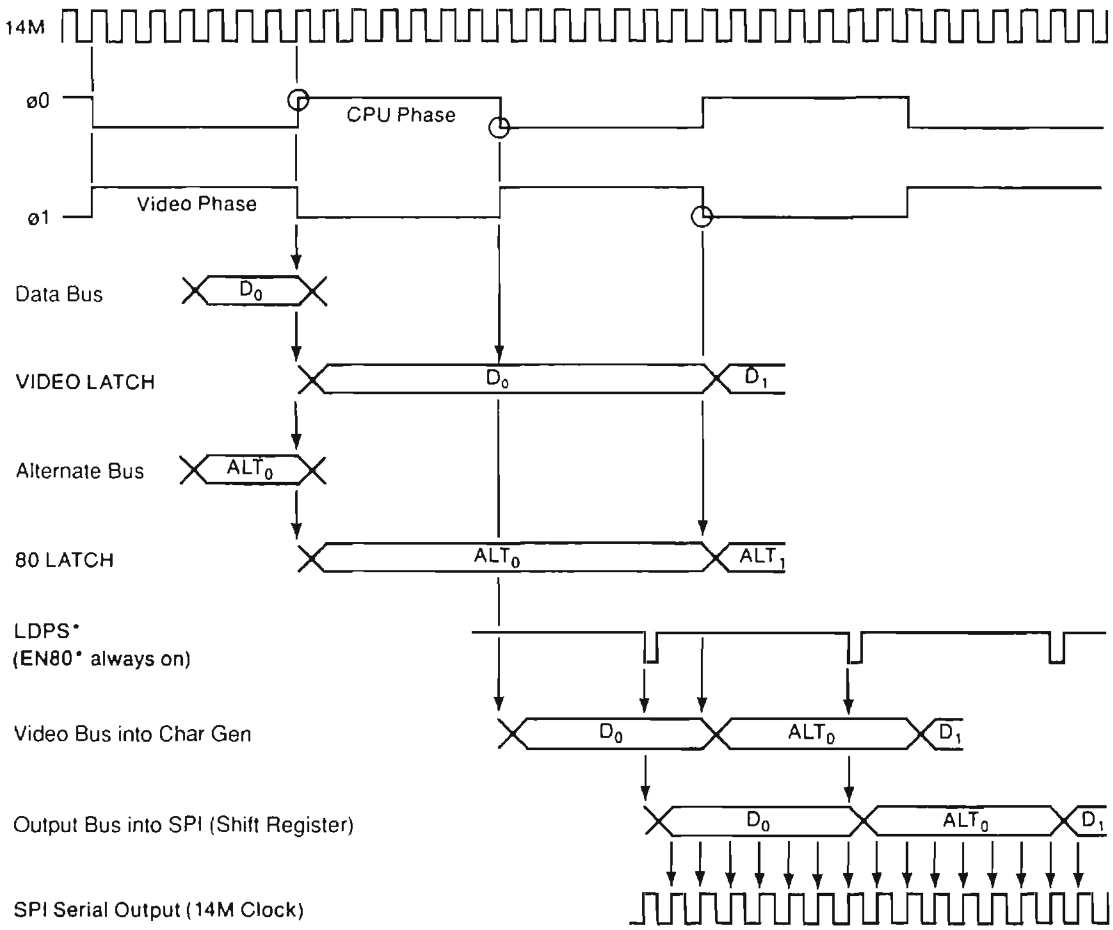
The bit patterns from the character generator are loaded into the 74166 parallel-to-serial shift register and output as a serial bit stream that goes to the video output circuit (see Figure 11-30). The shift register is controlled by signals LDPS' (load parallel-to-serial shifter) and VID7M (video 7 MHz). In 40-column mode, LDPS' strobes the output of the character generator into the shift register once each microsecond and VID7M shifts the bits out at 7 MHz (see Figure 11-31).

The addressing for the 80-column display is exactly the same as for the 40-column display: the 40 columns of display memory in auxiliary memory are addressed in parallel with the 40 columns in main memory. The data from these two memories reach the video data bus (lines VID0–VID7) via separate 74LS374 tri-state buffers. These buffers are loaded simultaneously (at the rising edge of PH0), but their outputs are sent to the character generator alternately by the falling edge of PH0 and PH1. In 80-column mode, LDPS' loads data from the character generator into the shift register twice during each microsecond, once during PH0 and once during PH1, and VID7M remains low, enabling the clock continuously at 14M (see Figure 11-32).

■ **Figure 11-31** 7 MHz video timing signals: 40-column, Lo-Res, and Hi-Res display



■ **Figure 11-32** 14 MHz video timing signals: 80-column and Double Hi-Res display



Lo-Res display

In the graphics modes, VA and VB are not used by the character generator, so the IOU uses lines SEGA and SEGB to transmit H0 and HIRES*, as shown in Table 11-31.

■ **Table 11-31** Character-generator control signals

IOU Pin	Iou signal name	Text mode	Graphics mode
3	SEGA	VA	H0
4	SEGB	VB	HIRES'
5	SEGC	VC	VC

The Lo-Res graphics display uses VC to divide the eight display lines corresponding to a row of characters into two groups of four lines each. Each row of data bytes is addressed eight times, the same as in Text mode, but each byte is interpreted as two nibbles. Each nibble selects 1 of 16 colors. During the upper four of the eight display lines, VC is low and the low-order nibble determines the color. During the lower four display lines, VC is high and the high-order nibble determines the color.

The bit patterns that produce the Lo-Res colors are read from the character-generator ROM in the same way the bit patterns for characters are produced in Text mode. The 74166 parallel-to-serial shift register converts the bit patterns to a serial bit stream for the video circuits (Figure 11-30).

The video signal generated by the Apple IIc family includes a short burst of 3.58 MHz signal that is used by an NTSC color monitor or color TV set to generate a reference 3.58 MHz color signal. The video signal of the Apple IIc-family computers produce color by interacting with this 3.58 MHz signal inside the monitor or TV set. Different bit patterns produce different colors by changing the duty cycles and delays of the bit stream relative to the 3.58 MHz color signal. To produce the small delays required for so many different colors, the shift register runs at 14 MHz and shifts out 14 bits during each cycle of the 1 MHz data clock. To generate a stream of 14 bits from each 8-bit pattern read from the ROM, the output of the shift register is connected back to the register's serial input to repeat the same 8 bits; the last 2 bits are ignored the second time around.

Each bit pattern is output for the same amount of time as a character: 1.02 microseconds. Because that is exactly enough time for three and a half cycles of the 3.58 MHz color signal, the phase relationship between the bit patterns and the signal changes by a half cycle for each successive pattern. To compensate for this, the character generator puts out one of two different bit patterns for each nibble, depending on the state of H0, the low-order bit of the horizontal counter.

Hi-Res display

The Hi-Res graphics pages begin at memory locations $2000 and $4000 (decimal 8192 and 16384). These page addresses are selected by address bits A13 and A14. In Hi-Res mode, these address bits are controlled by the Page2 and 80Store soft switches. As in Text mode, 80Store inhibits addressing of the second page because there is only one page of 80-column text available for mixed mode.

In Hi-Res graphics mode, the display data are still stored in blocks like the one shown in Figure 11-29, but there are eight of these blocks. As Tables 11-30 and 11-31 show, vertical counts VA, VB, and VC are used for address bits A10, A11, and A12, which address eight blocks of 1024 bytes each. Remember that in the display, VA, VB, and VC count adjacent horizontal lines in groups of eight. This addressing scheme maps each of those lines into a different 1024-byte block.

It might help to think of this scheme as a kind of eight-way multiplexer: it's as if eight text displays were combined to produce a single Hi-Res display, with each Text display providing one line of dots in turn, instead of a row of characters.

The Hi-Res bit patterns are produced by the character-generator ROM. In this mode, the bit patterns simply reproduce the 7 bits of display data. The low-order 6 bits of data reach the ROM via the video data bus VID0–VID5. The IOU sends the other 2 data bits to the ROM via RA9 and RA10.

The Hi-Res colors described in Chapter 6 are produced by the interaction between the video signal the bit patterns generate and the 3.58 MHz color signal generated inside the monitor or TV set. The Hi-Res bit patterns are always shifted out at 7 MHz, so each dot corresponds to a half-cycle of the 3.58 MHz color signal. Any part of the video signal that produces a single white dot between two black dots, or vice versa, is effectively a short burst of 3.58 MHz and is therefore displayed as color. In other words, a bit pattern consisting of alternating 1's and 0's gets displayed as a line of color. The Hi-Res graphics subroutines produce the appropriate bit patterns by masking the data bits with alternating 1's and 0's.

To produce different colors, the bit patterns must have different phase relationships to the 3.58 MHz color signal. If alternating 1's and 0's produce a certain color, say green, then reversing the pattern to 0's and 1's will produce the complementary color, purple. As in the Lo-Res mode, each bit pattern corresponds to three and a half cycles of the color signal, so the phase relationship between the data bits and the color signal changes by a half cycle for each successive byte of data. Here, however, the bit patterns produced by the hardware are the same for adjacent bytes; the color compensation is performed by the Hi-Res software, which uses different color masks for data being displayed in even and odd columns.

To produce other colors, bit patterns must have other timing relationships to the 3.58 MHz color signal. In Hi-Res mode, the Apple IIc family produces two more colors by delaying the output of the shift register by half a dot (70 ns), depending on the high-order bit of the data byte being displayed. (The high-order bit doesn't actually get displayed as a dot, because at 7 MHz there is only time to shift out seven of the eight bits.)

As each byte of data is sent from the character generator to the shift register, high-order data bit D7 is also sent to the TMG. If D7 is off, the TMG transmits shift-register timing signals LDPS* and VID7M normally. If D7 is on, the TMG delays LDPS* and VID7M by 70 nanoseconds, the time corresponding to half a dot. The bit pattern that formerly produced green now produces orange; the pattern for purple now produces blue.

◆ *Timing:* For 80-column text, the shift register is clocked at twice normal speed. When 80-column text is used with graphics in mixed mode, the TMG controls shift-register timing signals LDPS* and VID7M so that the graphics portion of the display works correctly even when the text window is in 80-column mode.

Double Hi-Res display

Double Hi-Res graphics mode displays 2 bytes in the time normally required for 1, but it uses Hi-Res graphics Pages 1 and 1X instead of Text and Lo-Res Pages 1 and 1X.

◆ *Note:* There is a second pair of bytes, HRP2 and HRP2X, which can be used to display a second Double Hi-Res page.

Double Hi-Res graphics mode displays each pair of data bytes as 14 adjacent dots, 7 from each byte. The high-order bit (color-select bit) of each byte is ignored. The auxiliary-memory byte is displayed first, so data from auxiliary memory appear in columns 0–6, 14–20, and so on, up to columns 547–552. Data from main memory appear in columns 7–13, 21–27, and so on, up to 553–559.

As in 80-column Text mode, there are twice as many dots across the display screen, so the dots are only half as wide. On a TV set or low-bandwidth (less than 14 MHz) monitor, single dots are dimmer than normal.

◆ *Note:* Except for some expensive color monitors, any video monitor with a bandwidth as high as 14 MHz is a monochrome monitor.

The main memory and auxiliary memory are connected to the address bus in parallel, so both are activated during the display cycle. The rising edge of PH0 clocks a byte of main memory data into the video latch, and a byte of auxiliary memory data into the 80 latch (Figure 11-30).

PH1 enables output from the (auxiliary) 80 latch, and PH0 enables output from the (main) video latch. Output from both latches goes to the character generator, where GR and SEGB* select Hi-Res graphics. LDPS* operates at 2 MHz in this mode, alternately gating the auxiliary byte and main byte into the parallel-to-serial shift register. VID7M is active (kept true) for Double Hi-Res display mode, so when it is ANDed with 14M, the result is still 14M. The 14M serial clock signal gates shift register output to VID, the video display hybrid circuit, for output to the display device.

Video output signals

The stream of video data generated by the display circuits described above goes to a hybrid circuit (VID) that adjusts the signals to the proper amplitudes and conditions the color burst.

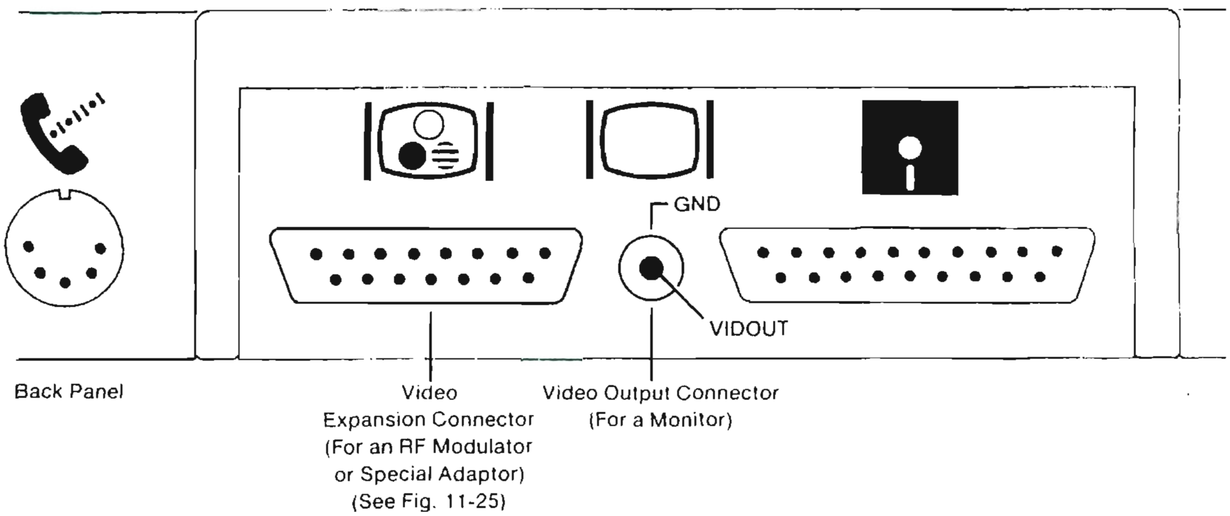
The resulting video signal is an NTSC-compatible composite-video signal that can be displayed on a standard video monitor. The signal is similar to the EIA (Electronic Industries Association) standard positive composite video. This signal is available in two places in Apple IIc family computers (see Figure 11-33):

- at the video output connector on the back of the Apple IIc-family computer
- at the video expansion connector (pin 12) on the back panel (see Table 11-32)

Monitor output

The sleeve of the video output connector at the center of the Apple IIc back panel is connected to ground and the tip is connected to the video output through a resistor network that attenuates it to about 1 volt and matches its impedance to 75 ohms. This arrangement is suitable for most video monitors.

■ **Figure 11-33** Video output back panel connectors



Video expansion output

The back panel of an Apple IIc family computer has a DB-15 connector for sophisticated video interfaces external to the computer.

Figure 11-34 shows the pin assignments for this connector; Table 11-32 describes the signals. In Table 11-32 the column labeled *Derived from* indicates what clock signals the video signals are derived from. LDPS', CREF, and PRAS have a maximum delay of 30 ns from the appropriate 14 MHz rising edge. SEROUT' is clocked out of a 74LS166 by the rising edge of 14M and has a maximum delay of 35 ns. VIDD7 is driven from a 74LS374 and has a maximum delay of 28 ns from the rising and (if 80-column) falling edges of PH1.

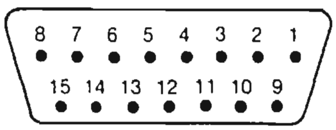
To align CREF so it is in the same phase at the beginning of every line, certain clock signals must be stretched. This stretch is for one 7M cycle (140 ns), and occurs at the end of each video line. All timing signals except 14M, 7M, and CREF are stretched.

▲ **Warning** The maximum allowable current drain of +12V regulated power at the video expansion connector is 300 milliamps. If the external device draws more than this, it can damage the computer or cause the power supply to shut down. ▲

▲ **Warning** The signals at the DB-15 on an Apple IIc are not the same as those at the DB-15 on the Apple IIGS. Do not attempt to plug a cable intended for one into the other. ▲

△ Important Several of the signals at the DB-15, such as 14 MHz, must be buffered within about 4 inches (10 cm) of the back panel connector—preferably inside a container directly connected to the back panel. For technical information, contact Apple Technical Support. △

■ **Figure 11-34** Video expansion connector pinouts



Pin	Signal	Pin	Signal
1	TEXT	9	PRAS*
2	14M	10	GR
3	SYNC*	11	SEROUT*
4	SEGB	12	NTSC
5	1VSOUND	13	GND
6	LDPS*	14	VIDD7
7	WNDW*	15	CREF
8	+12V		

■ **Table 11-32** Video expansion connector signals

Pin	Derived from	Signal	Description
1	PH0	TEXT	Video text signal from TMG; set to inverse of GR, except in Double Hi-Res mode
2		14M	14 MHz master timing signal from the system oscillator
3	Q3	SYNC*	Displays horizontal and vertical synchronization signal from IOU pin 39

■ **Table 11-32** Video expansion connector signals (continued)

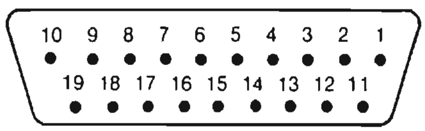
Pin	Derived from	Signal	Description
4	PRAS	SEGB	Displays vertical counter bit from IOU pin 4; in Text mode indicates second low-order vertical counter; in graphics mode indicates Lo-Res
5		1VSOUND	One-volt sound signal from pin 5 of the audio circuit (AUD)
6	14M	LDPS*	Video shift-register load enable from pin 12 of TMG
7	PRAS	WND**	Active area display blanking; includes both horizontal and vertical blanking
8		+12V	Regulated +12 volts DC; can drive 300 mA
9	14M	PRAS*	RAM row-address strobe from TMG pin 19
10	PRAS	GR	Graphics mode enable from IOU pin 2
11	14M	SEROUT*	Serialized character-generator output from pin 1 of the 74LS166 shift register
12		NTSC	Composite NTSC video signal from VID hybrid chip
13		GND	Ground reference and supply
14	PH0	VIDD7	From 74LS374 video latch; causes half-dot shift if high
15	14M	CREF	Color reference signal from TMG pin 3; 3.58 MHz

Disk I/O

Disk I/O for 5.25-inch drives is supported by the IWM disk controller unit. The 3.5-inch drives are supported through the IWM and the MIG. External drives are attached via a DB-19 connector. Figure 11-35 shows this connector. Table 11-33 describes the pin assignments for the disk connector on the original, UniDisk 3.5, and memory expansion versions of the Apple IIc. Table 11-34 describes the pin assignments for the disk connector on the Apple IIc Plus. Supply voltages come from the power supply; all other signals come from the IWM or MIG, as described in the sections "The Disk Controller Unit (IWM)" and "The Apple IIc Plus Multidrive Interface Glue (MIG)" earlier in this chapter.

▲ **Warning** The power supply available at this connector is for a Disk IIc, UniDisk 3.5, Apple 3.5, or similar type drive only. Do not use power from the external disk connector for any other purpose—you may damage the internal power supply. ▲

■ **Figure 11-35** External disk drive connector



Pin	Signal	Pin	Signal
1,2,3,4	GND	13	SEEKPH2
5	−12V	14	SEEKPH3
6	+5V	15	WRREQ*
7,8	+12V	16	N.C.
9	EXTINT*	17	DR2*
10	WRPROT	18	RDDATA
11	SEEKPH0	19	WRDATA
12	SEEKPH1		

■ **Table 11-33** External disk drive connector signals for the Apple IIc family

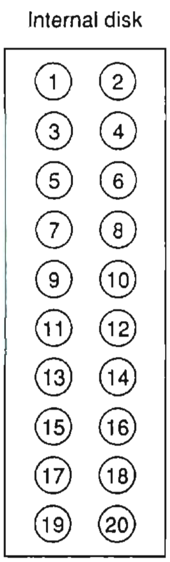
Connector pin	Signal	Description
1,2,3,4	GND	Ground
5	−12V	−12 volts
6	+5V	+5 volts
7,8	+12V	+12 volts
9	EXTINT*	External interrupt line
10	WRPROT	Write-protect input
11	SEEKPH0	Register select line CA0
12	SEEKPH1	Register select line CA1
13	SEEKPH2	Register select line CA2
14	SEEKPH3	Register write strobe LSTRB
15	WRREQ*	Write data request
17	EN2*	Drive 2 select
18	RDDATA	Read data input
19	WRDATA	Write data output

■ **Table 11-34** External disk drive connector signals for the Apple IIc Plus

Connector pin	Signal	Description
1,2,3	GND	Ground
4	3.5*	3.5-inch disk drive select
5	−12V	−12 volts
6	+5V	+5 volts
7,8	+12	+12 volts
9	ENBL*	Drive 2 enable
10	SENSE	Write-protect input
11	EXTSPH0	Register select line CA0
12	SEEKPH1	Register select line CA1
13	SEEKPH2	Register select line CA2
14	SEEKPH3	Register write strobe LSTRB
15	WRREQ*	Write data request
16	HDSEL2	Register select line SEL
17	EN1*	Drive 1 enable
18	RDDATA	Read data input
19	WRDATA	Write data output

Figure 11-36 shows the internal disk drive connector for all versions of the Apple IIc, including the Apple IIc Plus. Table 11-35 describes the pin assignments for the Apple IIc computers, and Table 11-36 describes the pin assignments for the Apple IIc Plus.

■ **Figure 11-36** Internal disk drive connector for the Apple IIc and Apple IIc Plus



■ **Table 11-35** Internal disk drive connector signals for the Apple IIc

Connector pin	Signal	Description
1	GND	Ground
2	SEEKPH0	Register select line CA0
3	GND	Ground
4	SEEKPH1	Register select line CA1
5	GND	Ground
6	SEEKPH2	Register select line CA2
7	GND	Ground
8	SEEKPH3	Register write strobe LSTRB
9	–12V	–12 volts
10	WRREQ*	Write request
11	+5V	+5 volts
12	+5V	+5 volts
13	+12V	+12 volts
14	EN1*	Drive 1 select
15	+12V	+12 volts
16	RDDATA	Read data input
17	+12V	+12 volts
18	WRDATA	Write data output
19	DISKACTY	Disk activity light
20	WRPROT	Write-protect input

■ **Table 11-36** Internal disk drive connector signals for the Apple IIc Plus

Connector pin	Signal	Description
1	GND	Ground
2	SEEKPH0	Register select line CA0
3	GND	Ground
4	SEEKPH1	Register select line CA1
5	GND	Ground
6	SEEKPH2	Register select line CA2
7	GND	Ground
8	SEEKPH3	Register write strobe LSTRB

■ **Table 11-36** Internal disk drive connector signals for the Apple IIc Plus (continued)

Connector pin	Signal	Description
9	EJECT	Disk eject
10	WRREQ'	Write data request
11	+5V	+5 volts
12	HDSEL1'	Register select line SEL
13	+12V	+12 volts
14	INTEN1'	Internal disk drive enable
15	+12V	+12 volts
16	RDDATA	Read data input
17	+12V	+12 volts
18	WRDATA	Write data output
20	CSOUT1'	Disk out signal

Serial I/O

Each Apple IIc–family computer has built into it two 6551 asynchronous communication interface adapters (ACIA) and supporting input and output buffers for full-duplex serial communication. Figure 12-37 is a block diagram of the Apple IIc family serial ports. ACIA outputs are buffered by a 1448 quad line driver, and ACIA inputs are buffered by a 1489 quad line receiver.

■ **Figure 11-37** Serial port circuits

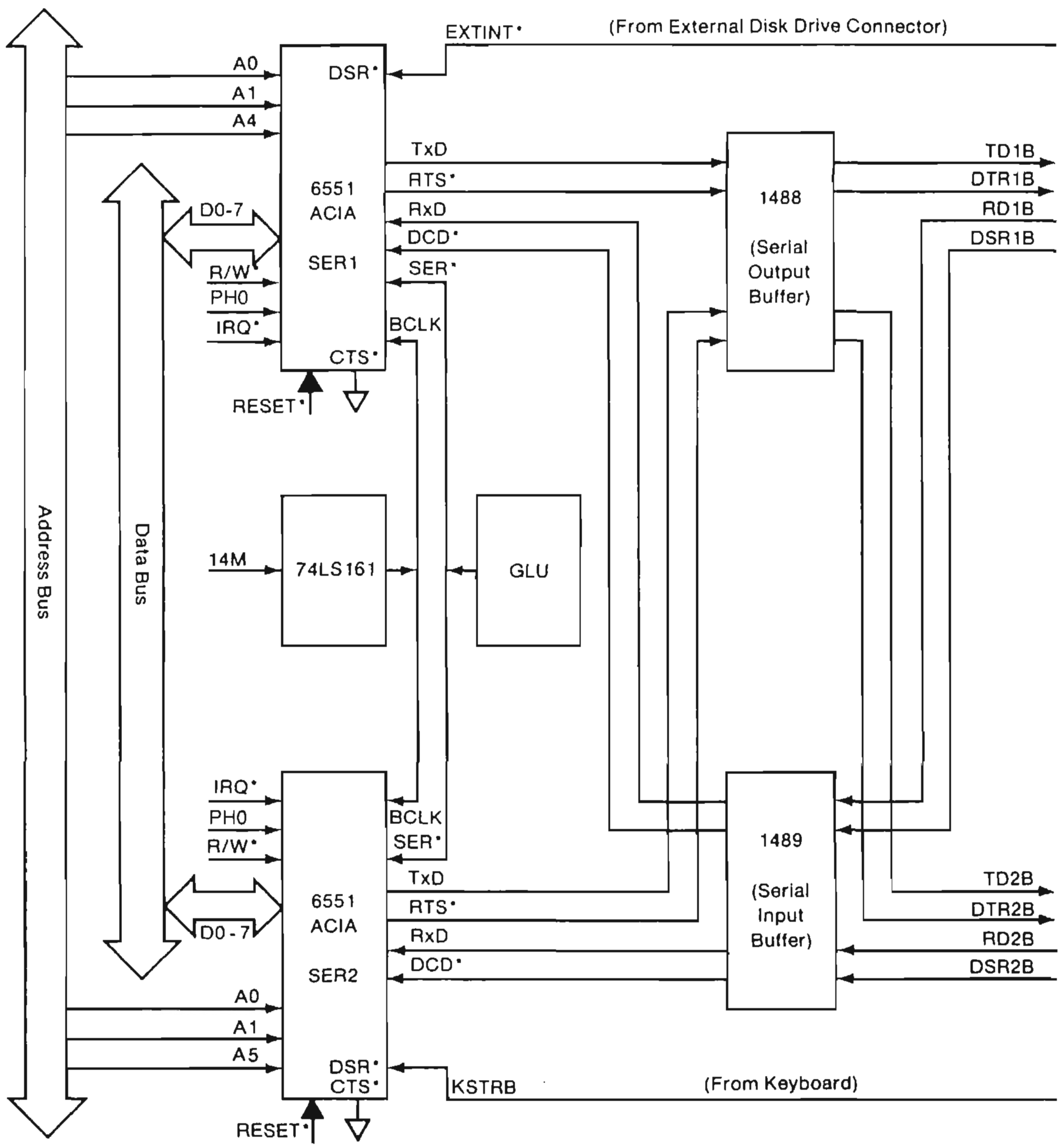
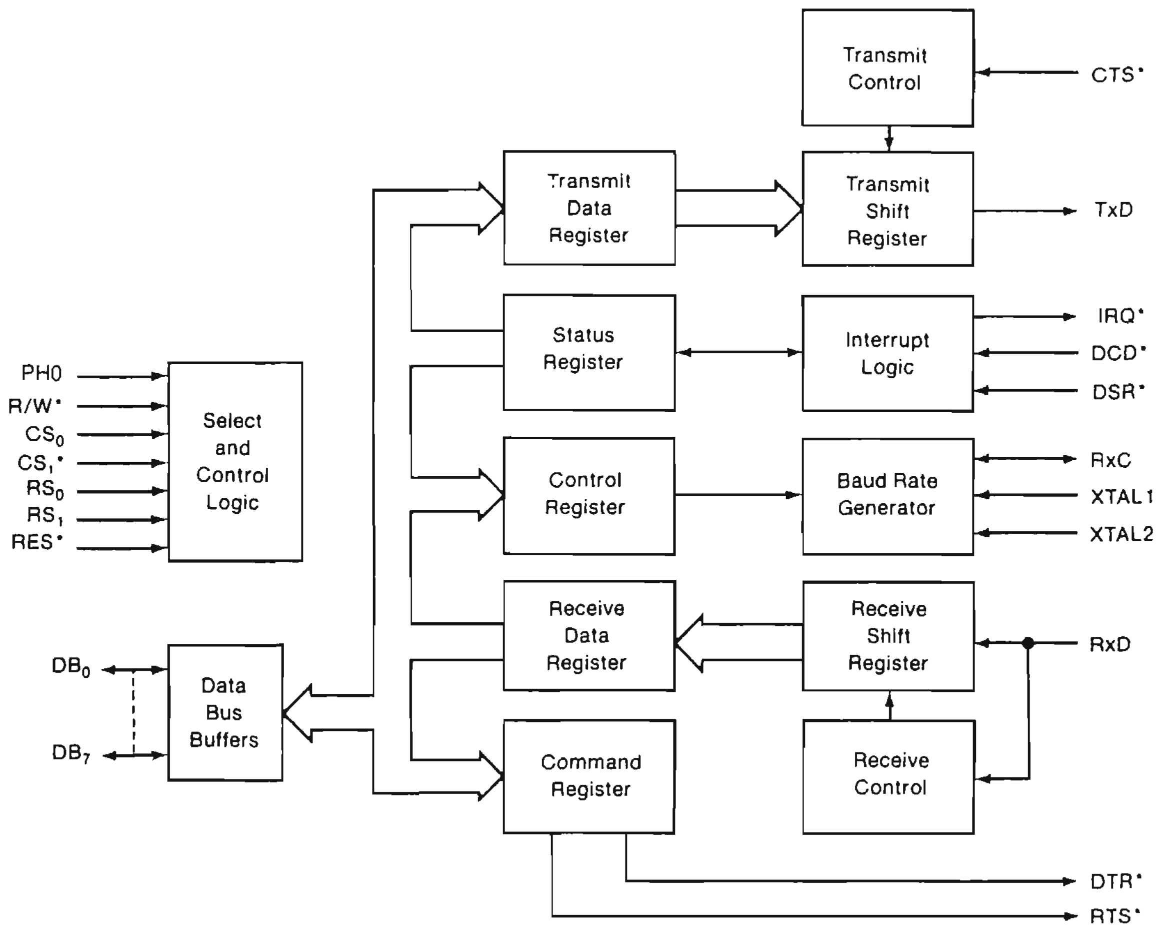


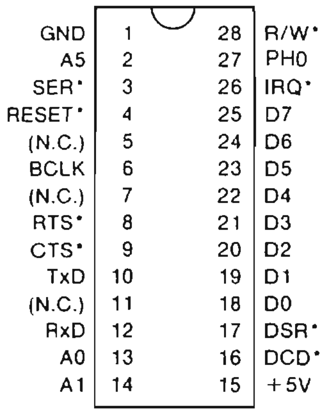
Figure 11-38 is a detailed block diagram of the 6551 ACIA. The registers are described later in this section.

■ **Figure 11-38** 6551 ACIA block diagram; (© 1978 by Synertek Inc.; used by permission)



The ACIA pin assignments are shown in Figure 11-39 and described in Table 11-37. Note that the two 6551 ACIA chips are not used in exactly the same way—each one supports a different set of interrupts.

■ **Figure 11-39** ACIA pinouts



In the Apple IIc, the port 1 6551 reads external interrupts (EXTINT*) from the external disk drive connector on the 6551 chip's Data Set Ready (DSR) pin. This input is tied to +5 volts through a 3.3-kilohm pullup resistor. The EXTINT* signal is not available on the Apple IIc Plus.

■ **Table 11-37** ACIA signal descriptions

Pin	Signal	Description
1	GND	Power and signal common ground
2	A4	Address line 4 to select serial port 1
	A5	Address line 5 to select serial port 2
3	SER*	Serial device select from GLU
4	RESET*	Resets both serial ports
5	N.C.	No connection
6	BCLK	Baud rate clock from 1.84 MHz oscillator
7	N.C.	No connection
8	RTS*	Request to Send line
9	CTS*	Clear to Send line (tied to ground)
10	TXD	Transmit Data line
11	N.C.	Not connected (Apple IIc)
	DTR	Data Terminal Ready (Apple IIc Plus)
12	RXD	Receive Data line
13,14	A0,A1	Address lines 0 and 1

■ **Table 11-37** ACIA signal descriptions (continued)

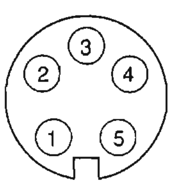
Pin	Signal	Description
15	+5V	+5 volt supply
16	DSR*	Data Set Ready line
17	EXTINT*	External interrupt (Apple IIc port 1)
	+5V	Connected to +5V through a 3.3 KΩ resistor (Apple IIc Plus port 1)
	KSTRB	Keyboard strobe input (port 2)
18–25	D0–D7	8-bit data bus
26	IRQ*	Interrupt Request line
27	PH0	Phase 0 clock pulse
28	R/W*	Read/write select input

The back panel connectors for both serial ports are identical. In the original, UniDisk 3.5, and memory expansion versions of the Apple IIc they are 5-pin DIN type connectors. In the Apple IIc Plus they are 8-pin mini-DIN type connectors. You can use the same cables with the serial connectors on the Apple IIc Plus and the Macintosh; unlike the Macintosh, however, the Apple IIc serial ports do not support AppleTalk.

The pin assignments for the original, UniDisk 3.5, and memory expansion versions are shown in Figure 11-40 and described in Table 11-38.

The pin assignments for the Apple IIc Plus are shown in Figure 11-41 and described in Table 11-39.

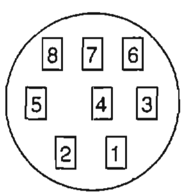
■ **Figure 11-40** Apple IIc serial port connectors



■ **Table 11-38** Apple IIc serial port connector signals

Pin	Signal	Description
1	DTR1B DTR2B	Data Terminal Ready output
2	TD1B TD2B	Transmit Data output
3	GND	Power and signal common
4	RD1B RD2B	Read Data input
5	DSR1B DSR2B	Data Set Ready input

■ **Figure 11-41** Apple IIc Plus serial port connectors



■ **Table 11-39** Apple IIc Plus serial port connector signals

Pin	Signal	Description
1	DTR1B DTR2B	Data Terminal Ready output
2	DSR1B DSR2B	Data Set Ready input
3	TD1B TD2B	Transmit Data output
4	GND	Power and signal common
5	RD1B RD2B	Read Data input
6	GND	Power and signal common

ACIA control register

Figure 11-42 shows the bit assignments for the ACIA control register, which the hardware locates at address $C09B for serial port 1, and $C0AB for serial port 2. This register determines the number of data and stop bits the ACIA transmits and receives, and the clock source and transmission rate (baud) to use for data transfer.

■ **Figure 11-42** ACIA control register (copyright © 1978 by Synertek Inc.; used by permission)

Port 1 = $C09B
Port 2 = $C0AB
Control Register

7	6	5	4	3	2	1	0

Stop Bits (bit 7)

0 = 1 stop bit
1 = 2 stop bits
1 stop bit if word length = 8 bits and parity**
1½ stop bits if word length = 5 bits and no parity

Word Length (bits 6, 5)

Bit 6	Bit 5	Data Word Length
0	0	8
0	1	7
1	0	6
1	1	5

Receiver Clock Source (bit 4)

0 = External receiver clock
1 = Baud rate generator

Baud Rate Generator (bits 3–0)

3	2	1	0	Baud Rate Generator
0	0	0	0	16x External Clock
0	0	0	1	50 baud
0	0	1	0	75
0	0	1	1	109.92
0	1	0	0	134.58
0	1	0	1	150
0	1	1	0	300
0	1	1	1	600
1	0	0	0	1200
1	0	0	1	1800
1	0	1	0	2400
1	0	1	1	3600
1	1	0	0	4800
1	1	0	1	7200
1	1	1	0	9600
1	1	1	1	19200

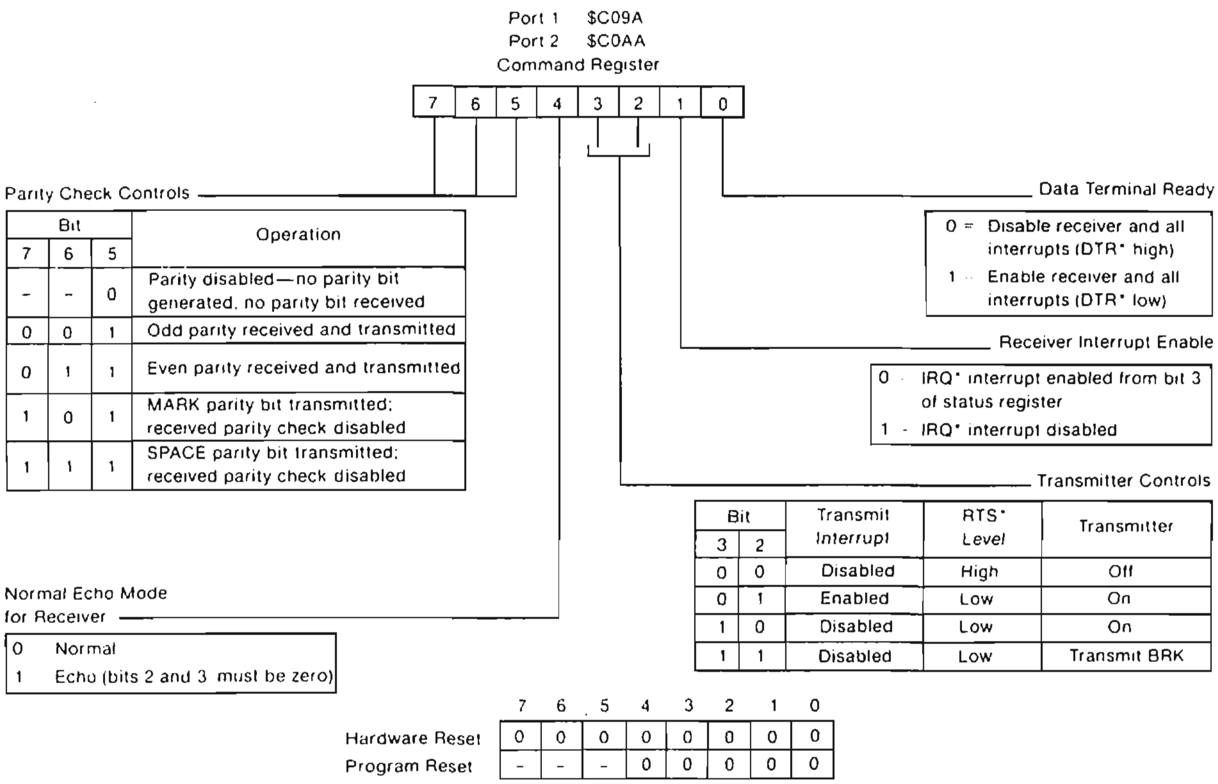
**This allows for 9-bit transmission (8 data plus parity).

	7	6	5	4	3	2	1	0
Hardware Reset	0	0	0	0	0	0	0	0
Program Reset	–	–	–	–	–	–	–	–

ACIA command register

Figure 11-43 shows the bit assignments for the ACIA command register, which the hardware locates at address \$C09A for serial port 1, and at \$C0AA for serial port 2. This register controls specific transmit and receive functions: parity checking, echoing input to output, allowing transmit and receive interrupts, and setting levels for the Data Terminal Ready and Request to Send signals.

■ **Figure 11-43** ACIA command register (copyright © 1978 by Synertek Inc.; used by permission)



Bit 7	Bit 6	Bit 5	Operation
-	-	0	Parity disabled—no parity bit generated, no parity bit received
0	0	1	Odd parity received and transmitted
0	1	1	Even parity received and transmitted
1	0	1	MARK parity bit transmitted; received parity check disabled
1	1	1	SPACE parity bit transmitted; received parity check disabled

Data Terminal Ready:

0 =	Disable receiver and all interrupts (DTR* high)
1 =	Enable receiver and all interrupts (DTR* low)

Receiver Interrupt Enable:

0 -	IRQ* interrupt enabled from bit 3 of status register
1 -	IRQ* interrupt disabled

Transmitter Controls:

Bit 3	Bit 2	Transmit Interrupt	RTS* Level	Transmitter
0	0	Disabled	High	Off
0	1	Enabled	Low	On
1	0	Disabled	Low	On
1	1	Disabled	Low	Transmit BRK

Normal Echo Mode for Receiver:

0	Normal
1	Echo (bits 2 and 3 must be zero)

	7	6	5	4	3	2	1	0
Hardware Reset	0	0	0	0	0	0	0	0
Program Reset	-	-	-	0	0	0	0	0

ACIA status register

Figure 11-44 shows the bit assignments for the ACIA status register, which is hard-wired to address \$C099 for serial port 1, and to \$C0A9 for serial port 2. This register reports the condition of the transmit/receive register, errors detected during data transfer, and the level of the Data Carrier Detect, Data Set Ready, and Interrupt Request lines.

■ **Figure 11-44** ACIA status register (© 1978 by Synertek Inc.; used by permission)

Port 1 = $C099
Port 2 = $C0A9

Bit	Status	Set By	Cleared By
0	Parity error†	0 = No error 1 = Error	Self-clearing**
1	Framing error†	0 = No error 1 = Error	Self-clearing**
2	Overrun†	0 = No error 1 = Error	Self-clearing**
3	Receive Data Register full	0 = Not full 1 = Full	Reading receive data register
4	Transmit Data Register empty	0 = Not empty 1 = Empty	Writing to transmit data register
5	DCD*	0 = DCD* low 1 = DCD* high	Not resettable; reflects DCD* state
6	DSR*	0 = DSR* low 1 = DSR* high	Not resettable; reflects DSR* state
7	IRQ	0 = No interrupt 1 = Interrupt	Reading status register

† No interrupt generated for these conditions.
** Cleared automatically after a read of RDR and the next error-free receipt of data.

	7	6	5	4	3	2	1	0
Hardware Reset	0	0	0	0	0	0	0	0
Program Reset	–	–	–	–	–	–	–	–

ACIA transmit/receive register

Each ACIA uses the same address—$C098 for serial port 1, $C0A8 for serial port 2—as temporary storage for both transmission and reception of data.

When the register is used for transmitting data, bit 0 is the leading bit to be transmitted; unused data bits are the high-order bits, which are ignored.

When the register is used for receiving data, bit 0 is the first bit received; unused data bits are the high-order bits, which are set to 0. Parity bits never appear in the receive data register; they are stripped off after being used for external parity checking.

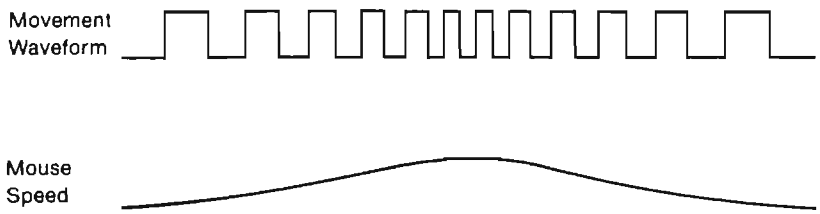
Mouse input

This section describes how mouse movement and direction are detected and interpreted by the Apple IIc family.

The mouse has a ball inside its housing that protrudes a small distance so that it turns when the mouse is pushed across a flat surface. Two wheels inside the housing, set at 90-degree angles to each other, are turned by the ball causing two disks to rotate. The disks have rectangular holes arranged near their edges, making them resemble the circular slide mounts used with stereoscopic slide viewers.

The light from a tiny infrared emitter reaches a photoreceptor whenever one of the holes on the disk lies between them. An internal circuit in the mouse causes the resulting voltage to swing quickly to a 1 or a 0 value as soon as a certain threshold is crossed. The result is something approximating a square wave (see Figure 11-45) that varies directly with the speed of mouse movement. One of the square waves generated in this way indicates the X component (X0) of mouse movement; the other, the Y component (Y0).

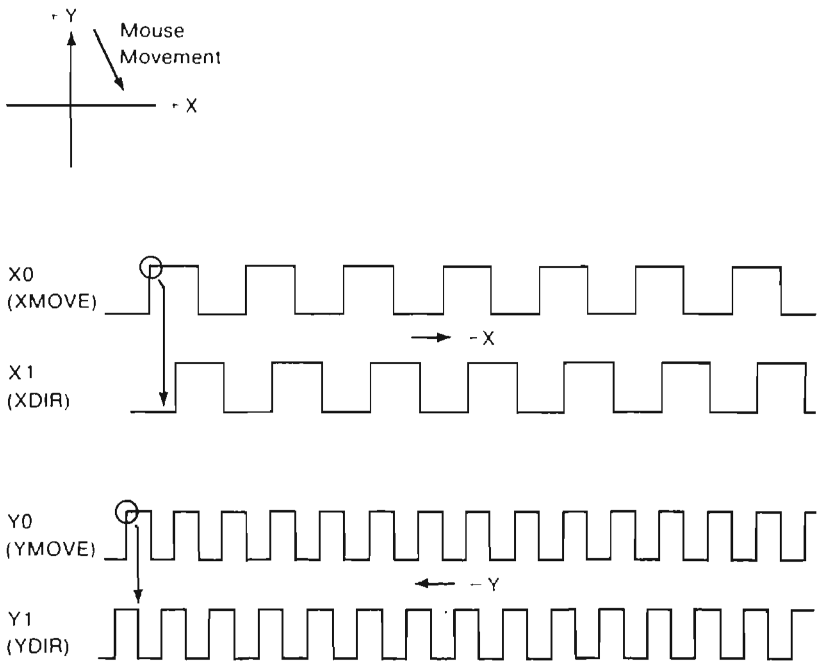
■ **Figure 11-45** Sample mouse waveform



Under program control, either the rising edge or the falling edge of each square wave can cause an interrupt, which the firmware handles by updating a counter. However, the program needs to know whether to add or to subtract 1 from a counter; that is, it needs to know the direction of X or Y movement.

There is a second infrared emitter/photoreceptor pair almost 180 degrees opposite the first pair for each disk. These pairs are positioned in such a way that the square waves they generate are approximately a quarter-wave offset from their respective movement waves (see Figure 11-46). These waveforms are called *X1* (X direction) and *Y1* (Y direction).

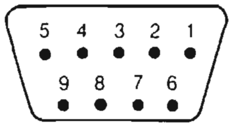
■ **Figure 11-46** Mouse movement and direction waveforms



When a rising edge of X0 causes an interrupt, a mouse-driver program can immediately check whether X1 is 0 (indicating a movement to the right) or 1 (indicating a movement to the left). Similarly, the mouse driver can read Y1 immediately after a Y0 interrupt to determine whether the mouse moved up or down one count along the Y axis.

Figure 11-47 shows the pin assignments for the mouse DB-9 connector on the back panel. Table 11-40 gives the signal names and descriptions.

■ **Figure 11-47** Mouse connector



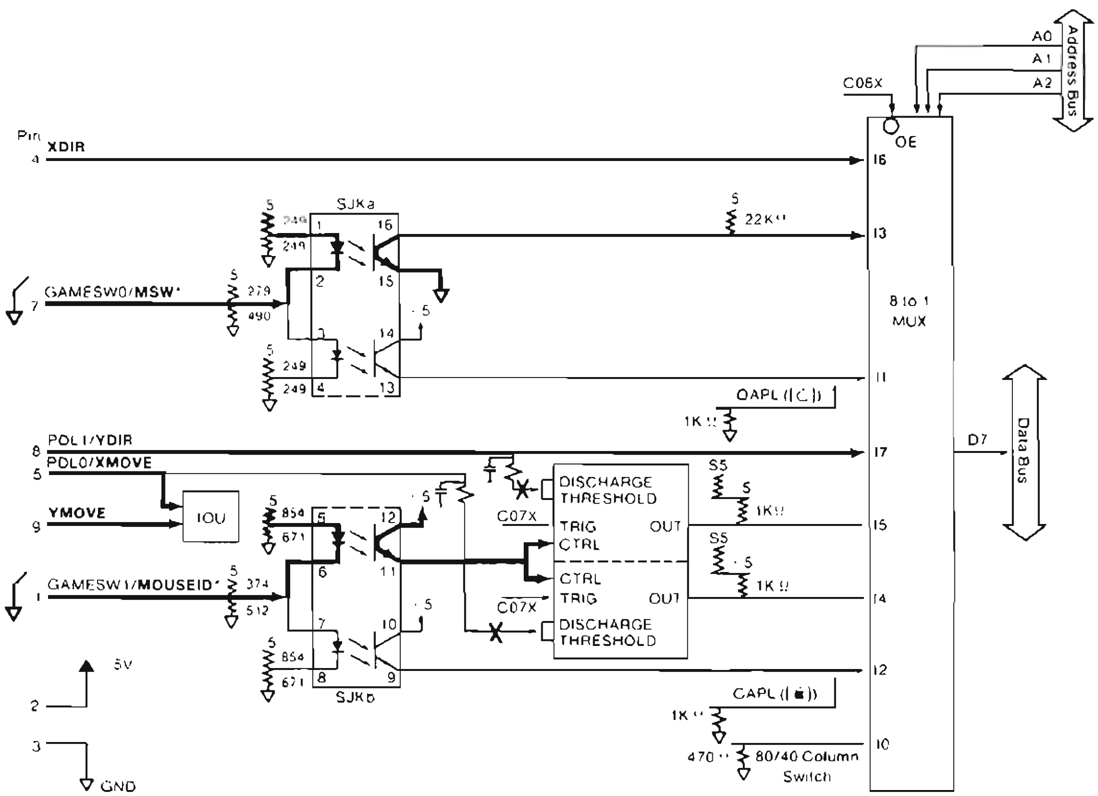
Pin	Signal
1	MOUSEID*
2	+5V
3	GND
4	XDIR
5	XMOVE
6	(N.C.)
7	MSW*
8	YDIR
9	YMOVE

■ **Table 11-40** Mouse connector signals

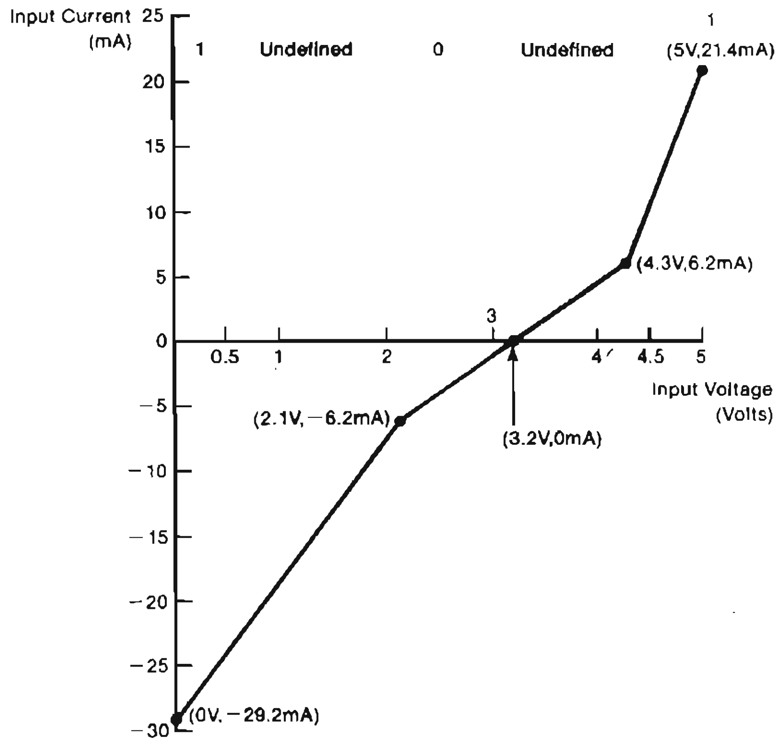
Pin	Signal	Description
1	MOUSEID*	Mouse identifier: when active, disables NE556 hand control timer
2	+5V	Total current drain from this pin must not exceed 100 mA
3	GND	System ground
4	XDIR	Mouse X-direction indicator
5	XMOVE	Mouse X-movement interrupt
6	N.C.	No connection
7	MSW*	Mouse button
8	YDIR	Mouse Y-direction indicator
9	YMOVE	Mouse Y-movement interrupt

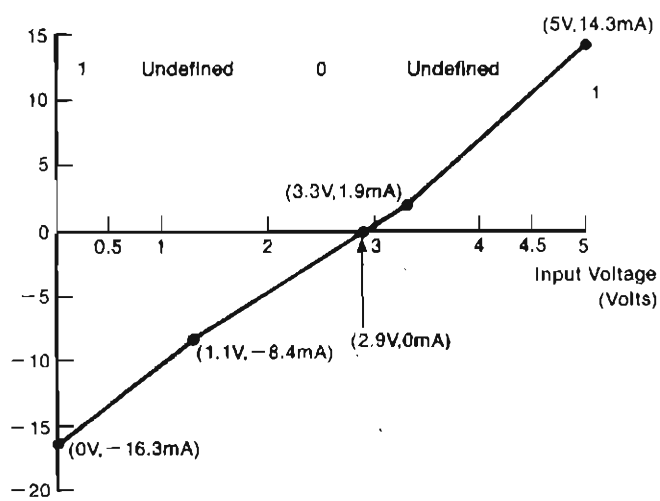
Figure 11-48 shows the mouse and hand control circuitry with the mouse circuits emphasized. Figure 11-49 illustrates the values of the mouse-button circuit when the button is pressed or not pressed. Pressing the button disables the NE556 by pulling the reset comparator threshold value up so that it cannot reset the flip flop. As a result the mouse-button input value remains at a TTL level.

■ **Figure 11-48** Mouse circuits



■ **Figure 11-49** Mouse button signals



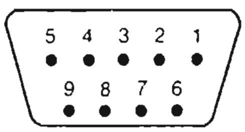


Hand control input

Several input signals that are individually controlled via soft switches are collectively referred to as the hand control (game) signals. These signals arrive in an Apple IIc family computer via the same DB-9 connector as the one used for the mouse, but an Apple IIc-family computer interprets these signals differently.

The DB-9 connector pin assignments and signal descriptions, as used for hand control input, appear in Figure 11-50 and Table 11-41.

■ **Figure 11-50** Hand control connector



Pin	Signal
1	GAMESW1
2	+5V
3	GND
4	Not used for hand controllers
5	PDL0
6	(N.C.)
7	GAMESW0
8	PDL1
9	Not used for hand controllers

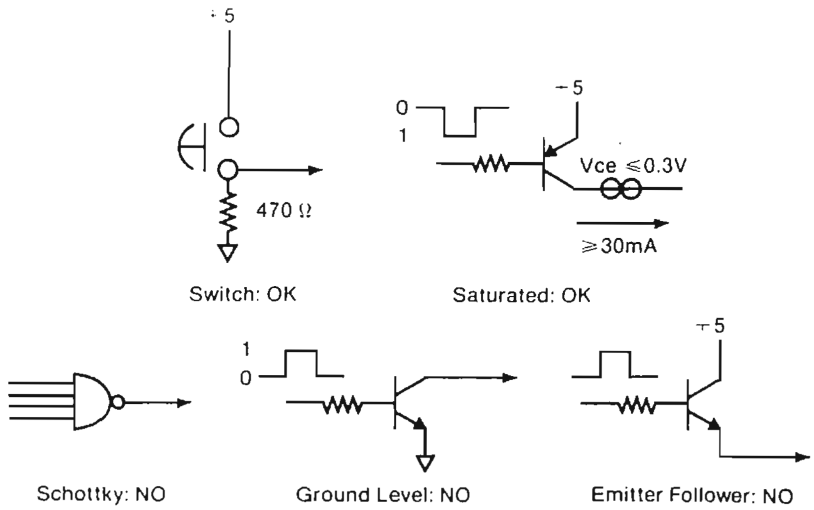
Even though they are normally used for hand controls, these signals can be used for other simple I/O applications. There are two 1-bit switch inputs, labeled *Sw0* and *Sw1*, and two analog inputs, called *paddles* and labeled *Pdl0* and *Pdl1*. Figure 11-51 shows how to connect the 1-bit switch inputs for compatibility with all other Apple II–series computers.

The switch inputs are multiplexed by a 74LS251 8-1 multiplexer enabled by the C06X' signal from the MMU. Depending on the low-order address, the appropriate game input is connected to bit 7 of the data bus. Figure 11-52 shows the mouse and hand control circuitry with the hand control circuits highlighted. Figure 11-53 illustrates the values of the hand control switch inputs when the switch is open or closed.

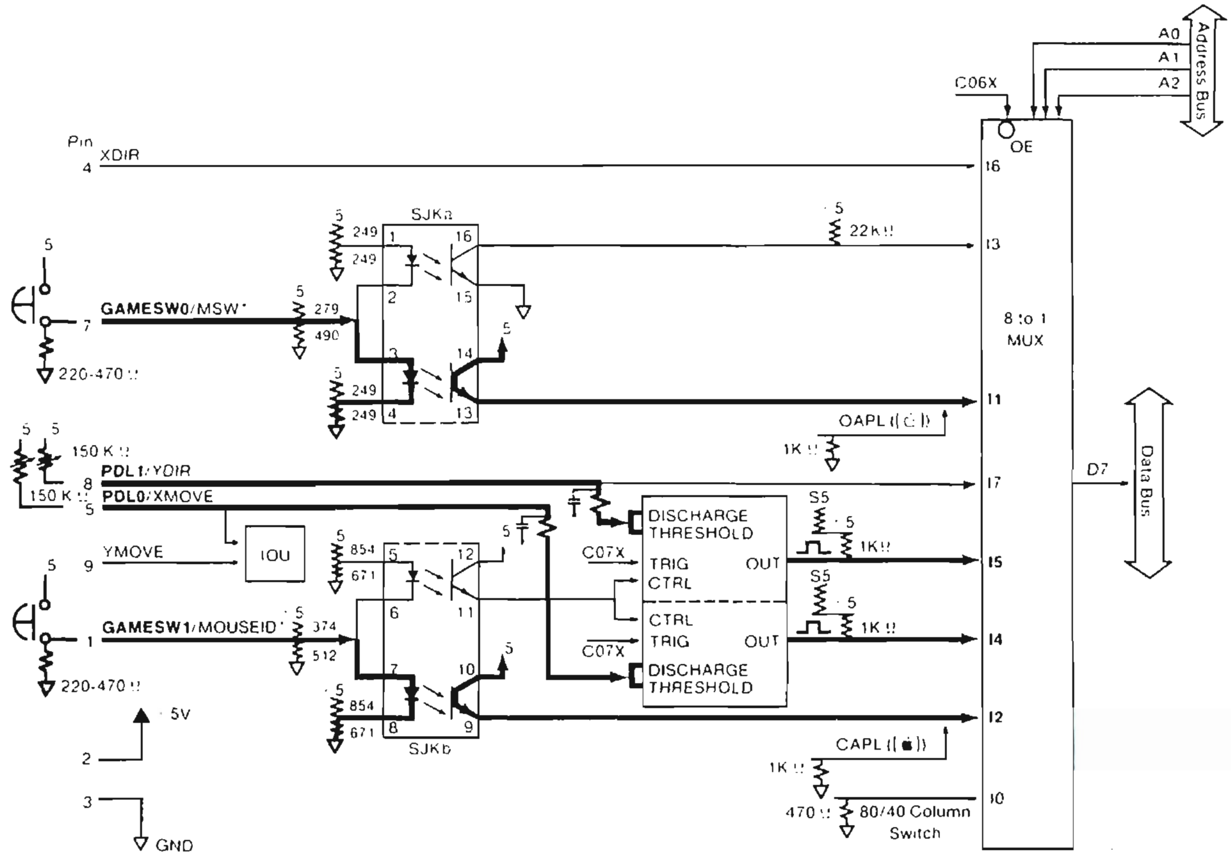
■ **Table 11-41** Hand control connector signals

Pin	Signal	Description
1	GAMESW1	Switch input 1 (sometimes called *paddle button 1*)
2	+5V	+5V power supply; total current drain from this pin must not exceed 100 mA
3	GND	System ground
4,9		Not used for hand controls
5,8	PDL0 and PDL1	Hand control inputs; each of these must be connected to a 150 KΩ variable resistor connected to +5V
6	N.C.	No connection
7	GAMESW0	Switch input 0 (sometimes called *paddle button 0*)

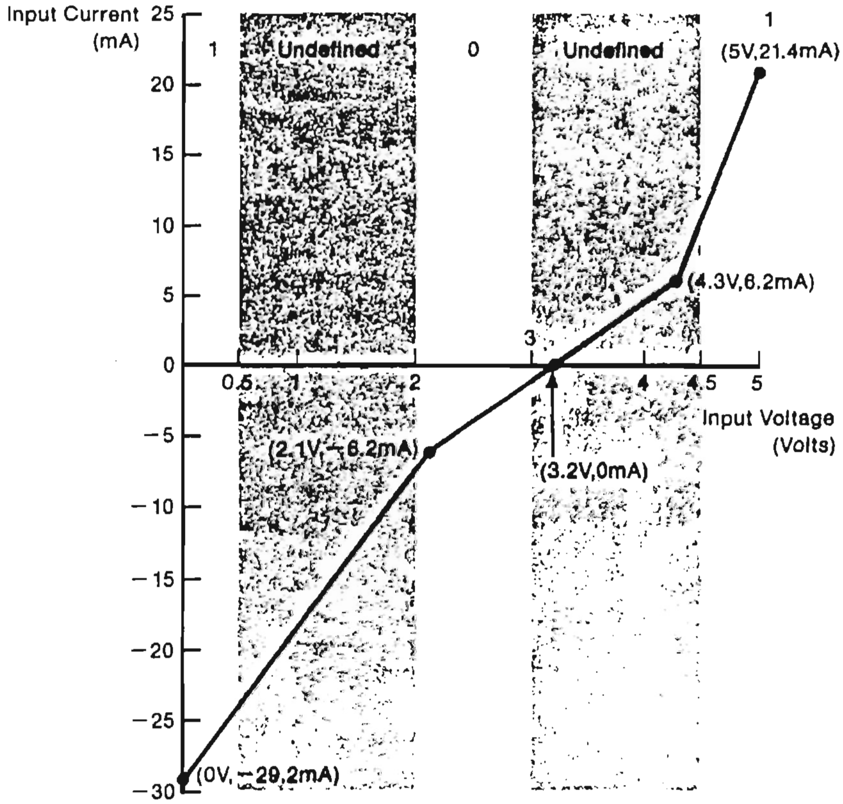
■ **Figure 11-51** How to connect switch inputs

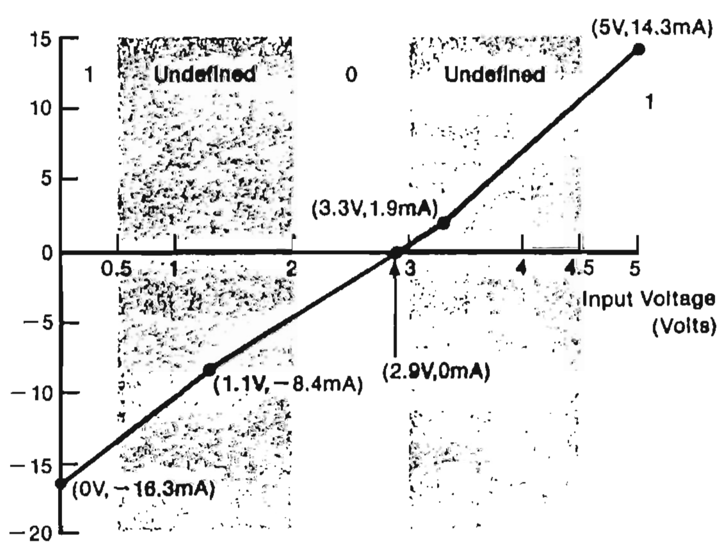


■ **Figure 11-52** Hand control circuits



■ **Figure 11-53** Hand control signals





The hand control inputs are connected to the timing inputs of an NE556 dual analog timer. Addressing $C07X sends a signal from MMU pin 22 that resets both timers and causes their outputs to go to 1 (high). A variable resistance of up to 150 kilohm connected between one of these inputs and the +5 volt supply controls the charging time of one of the two 0.022 microfarad capacitors.

When the voltage on the capacitor passes a certain threshold, the output of the NE556 changes back to 0 (low). Programs can determine the setting of a variable resistor by resetting the timers and then counting time until the selected timer input changes from high to low. The resulting count is proportional to the resistance.

△ Important The only way to ensure correct paddle values is to make sure the output of the paddle you intend to read is low before you trigger the timer. Triggering the timer starts the charging cycle for the capacitor in each paddle circuit; the cycle for one may not be completed by the time you have read the other. If you retrigger or read the other paddle too soon (that is, in less than 3 ms), you get a false value for it. △

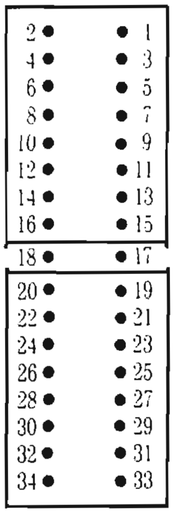
Memory expansion card

Memory expansion card I/O is supported by firmware and by an internal connector mounted on the main logic board in the memory expansion Apple IIc and the Apple IIc Plus. The connector pins are identified in Figure 11-54 and Table 11-42.

△ Apple IIc Plus Due to the faster clock rate used in the Apple IIc Plus computer, the memory expansion card manufactured by Apple Computer, Inc., for use in the memory expansion Apple IIc does not work in the Apple IIc Plus. To add a memory expansion card to the Apple IIc Plus, you must purchase a card made by another manufacturer specifically for the Apple IIc Plus. However, the RAM chips on an Apple Memory Expansion Card can be used on a memory expansion card in the Apple IIc Plus computer. △

For information on the Apple IIc Memory Expansion Card, refer to the *Apple IIc Memory Expansion Card Technical Reference*.

■ **Figure 11-54** Memory expansion card connector pinouts



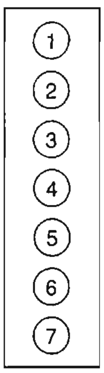
Pin	Signal	Pin	Signal
1	D0	18	A9
2	D1	19	A10
3	D2	20	A11
4	D3	21	A12
5	D4	22	A13
6	D5	23	A15
7	D6	24	A15
8	D7	25	$\overline{\text{RESET}}$
9	GND	26	R$\overline{\text{W}}$
10	GND	27	+5V
11	A0	28	+5V
12	A1	29	PHO
13	A4	30	GND
14	A5	31	7M
15	A6	32	GND
16	A7	33	Q3
17	A8	34	+5V

■ **Table 11-42** Memory expansion card connector signals

Pin	Signal	Description
1	D0	System data line 0
2	D1	System data line 1
3	D2	System data line 2
4	D3	System data line 3
5	D4	System data line 4
6	D5	System data line 5
7	D6	System data line 6
8	D7	System data line 7
9	GND	Ground
10	GND	Ground
11	A0	System address line 0
12	A1	System address line 1
13	A4	System address line 4
14	A5	System address line 5
15	A6	System address line 6
16	A7	System address line 7
17	A8	System address line 8
18	A9	System address line 9
19	A10	System address line 10
20	A11	System address line 11
21	A12	System address line 12
22	A13	System address line 13
23	A14	System address line 14
24	A15	System address line 15
25	RESET•	System reset
26	R/W•	Read/write enable
27	+5V	+5 volts
28	+5V	+5 volts
29	PH0	System clock
30	GND	Ground
31	7M	7 MHz clock
32	GND	Ground
33	Q3	Q3 clock
34	+5V	+5 volts

The Apple IIc Plus computer has an additional internal expansion connector to provide additional signals for memory expansion cards. The pin assignments for this connector are shown in Figure 11-55, and the signals are described in Table 11-43.

■ **Figure 11-55** Apple IIc Plus expansion connector



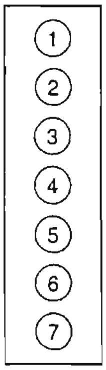
■ **Table 11-43** Apple IIc Plus expansion connector signals

Pin	Signal	Description
1	RA14	ROM address 14
2	INH*	Memory inhibit line
3	EN80*	Enable auxiliary RAM
4	ROMEN1*	Enable ROM
5	A3	System address bit 3
6	A2	System address bit 2
7	BUSEN*	Indicates valid access to MIG address space

Apple IIc Plus internal modem connector

The Apple IIc Plus computer has an internal connector to support modems. The pin assignments for this connector are shown in Figure 11-56, and the signals are described in Table 11-44.

■ **Figure 11-56** Apple IIc Plus internal modem connector



■ **Table 11-44** Apple IIc Plus internal modem connector signals

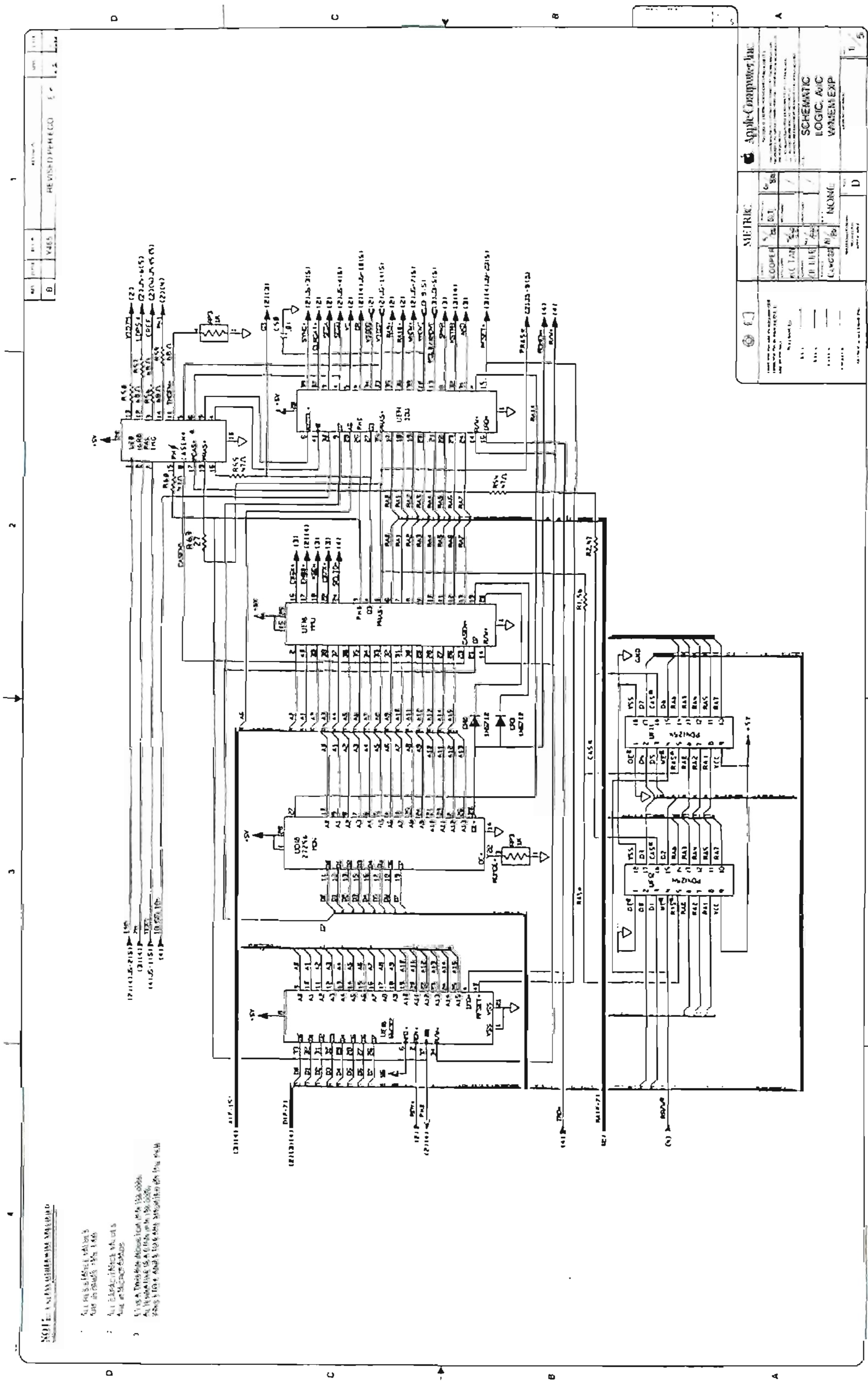
Pin	Signal	Description
1	-5V	-5 volts
2	RXD	Receive Data
3	TXD	Transmit Data
4	DCD	Data Carrier Detect
5	DTR	Data Terminal Ready
6	DSR	Data Signal Ready
7	GND	Ground

Note: All signals except -5V and GND are TTL levels

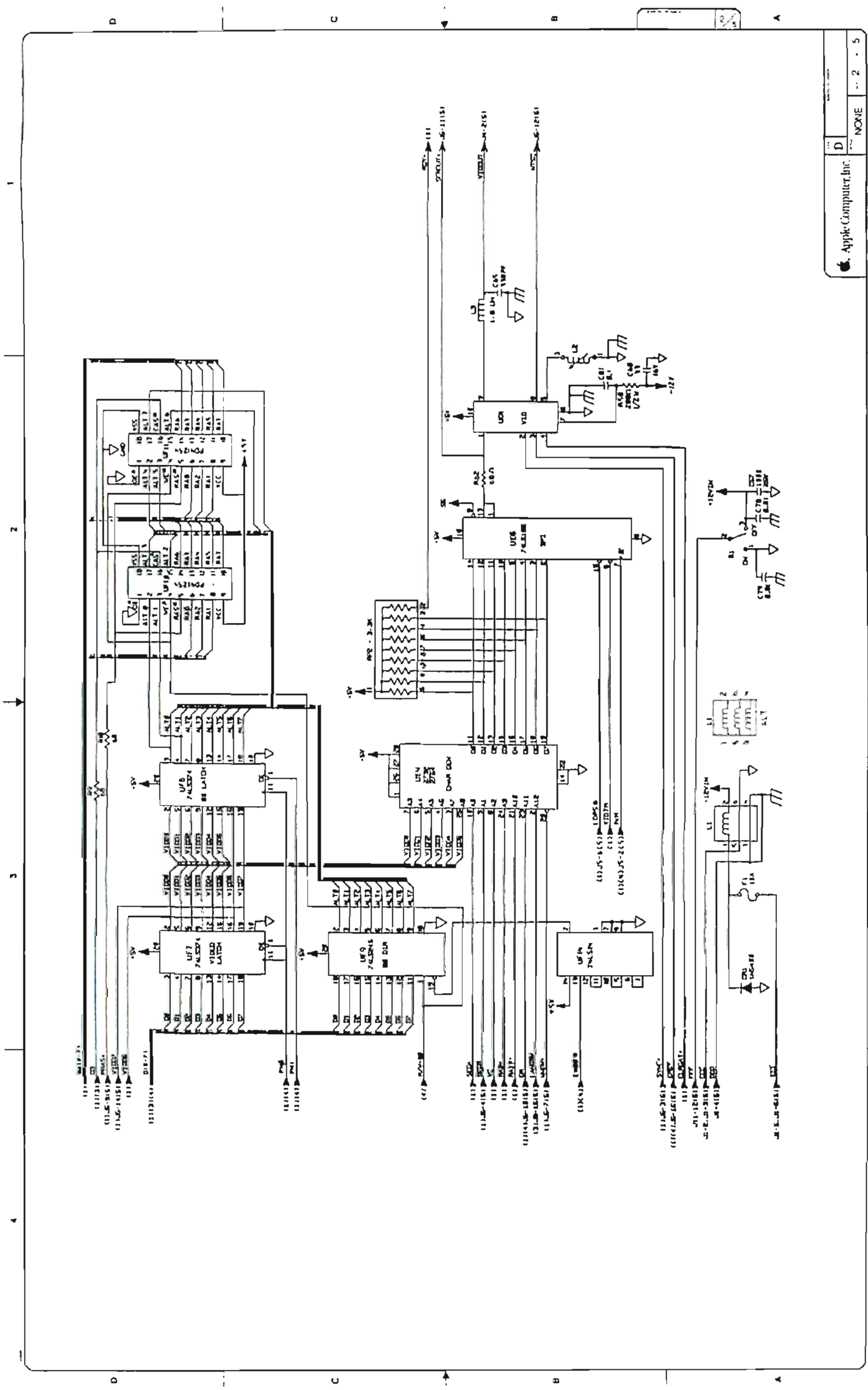
Schematic diagrams

Figure 11-57, on the following pages, is a set of schematic diagrams for the Apple IIc. Figure 11-58 is a set of schematic diagrams for the Apple IIc Plus.

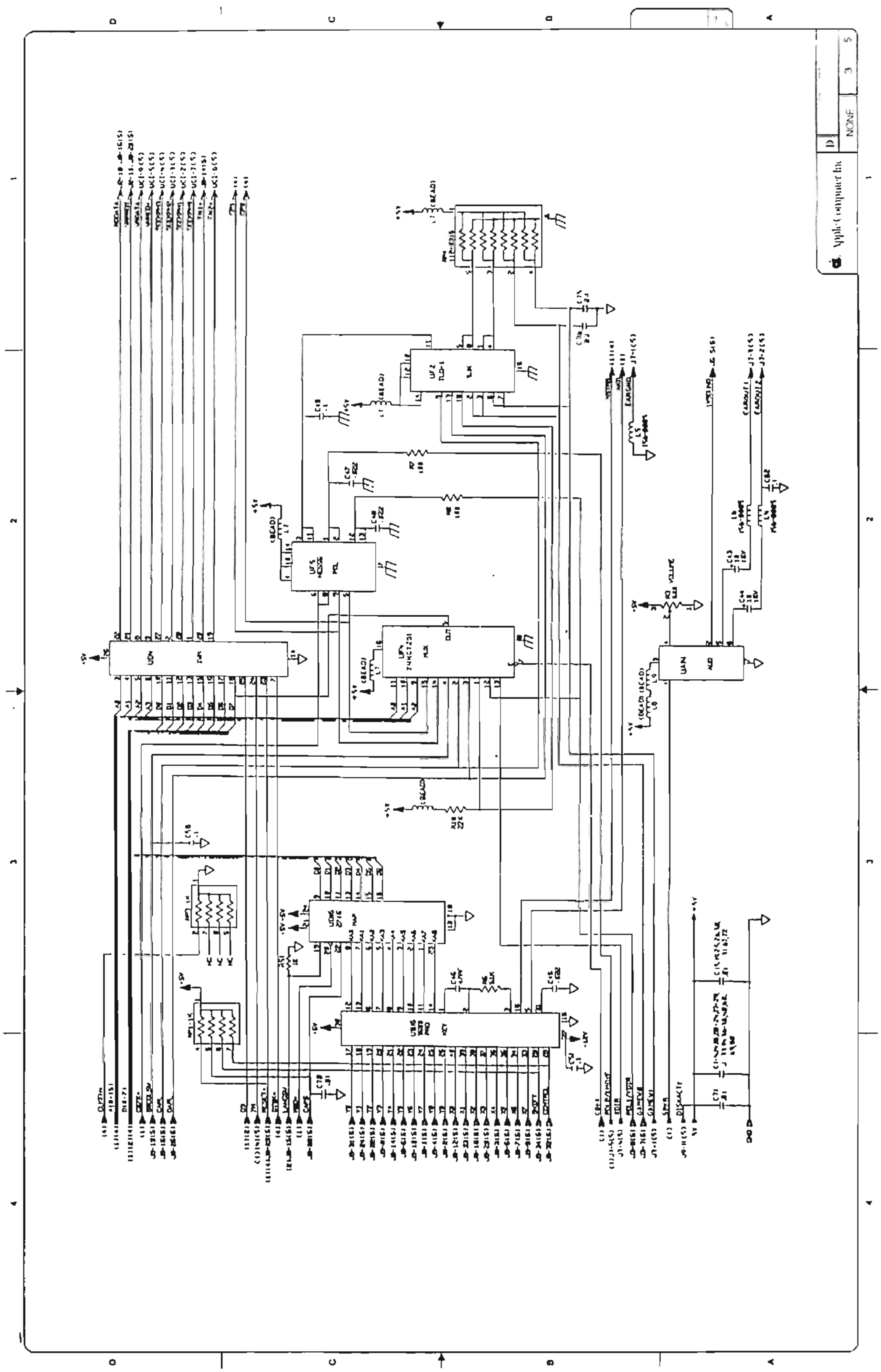
■ **Figure 11-57** Schematic diagrams for the Apple IIc



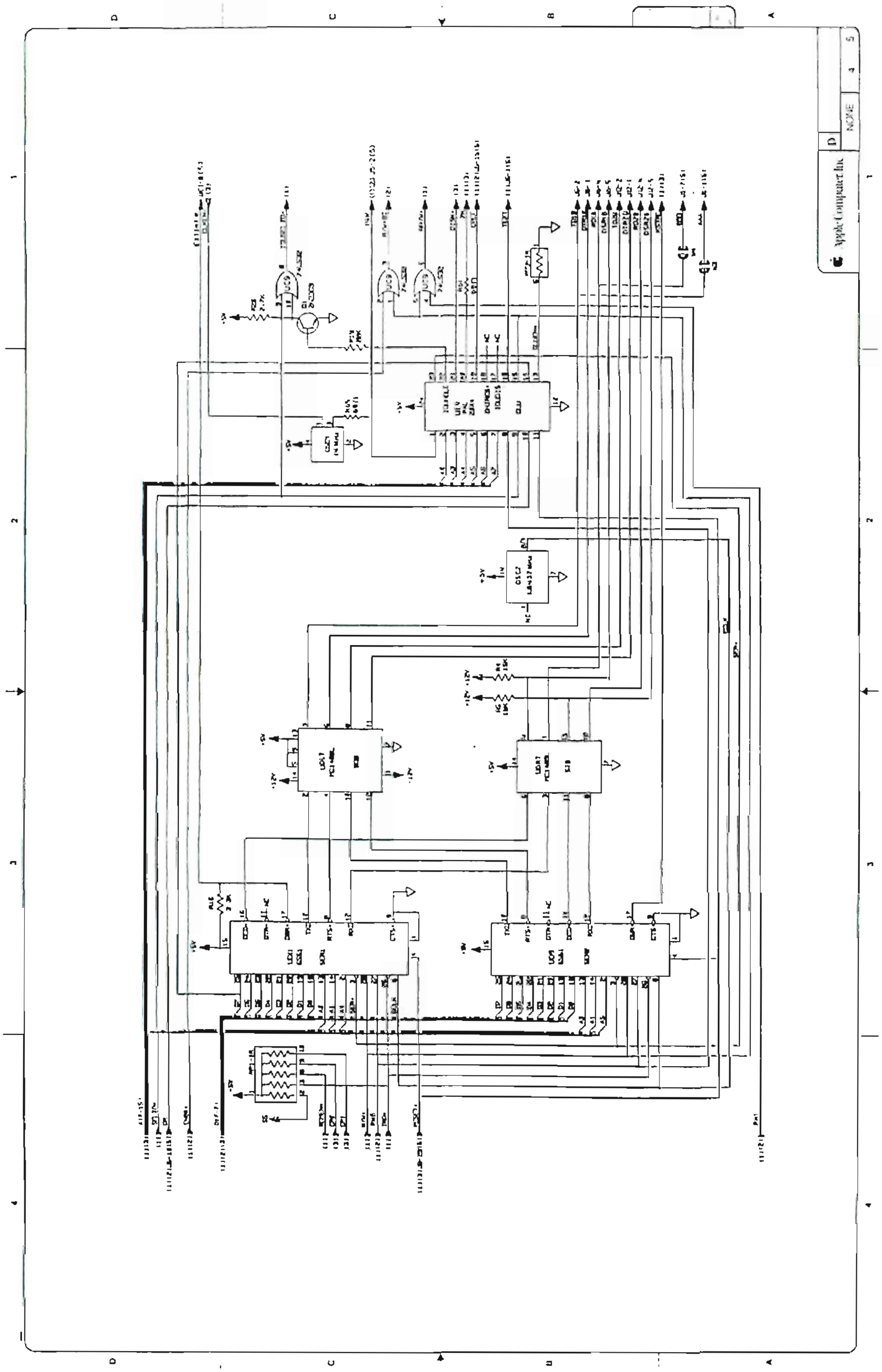
■ **Figure 11-57** Schematic diagrams for the Apple IIc (continued)



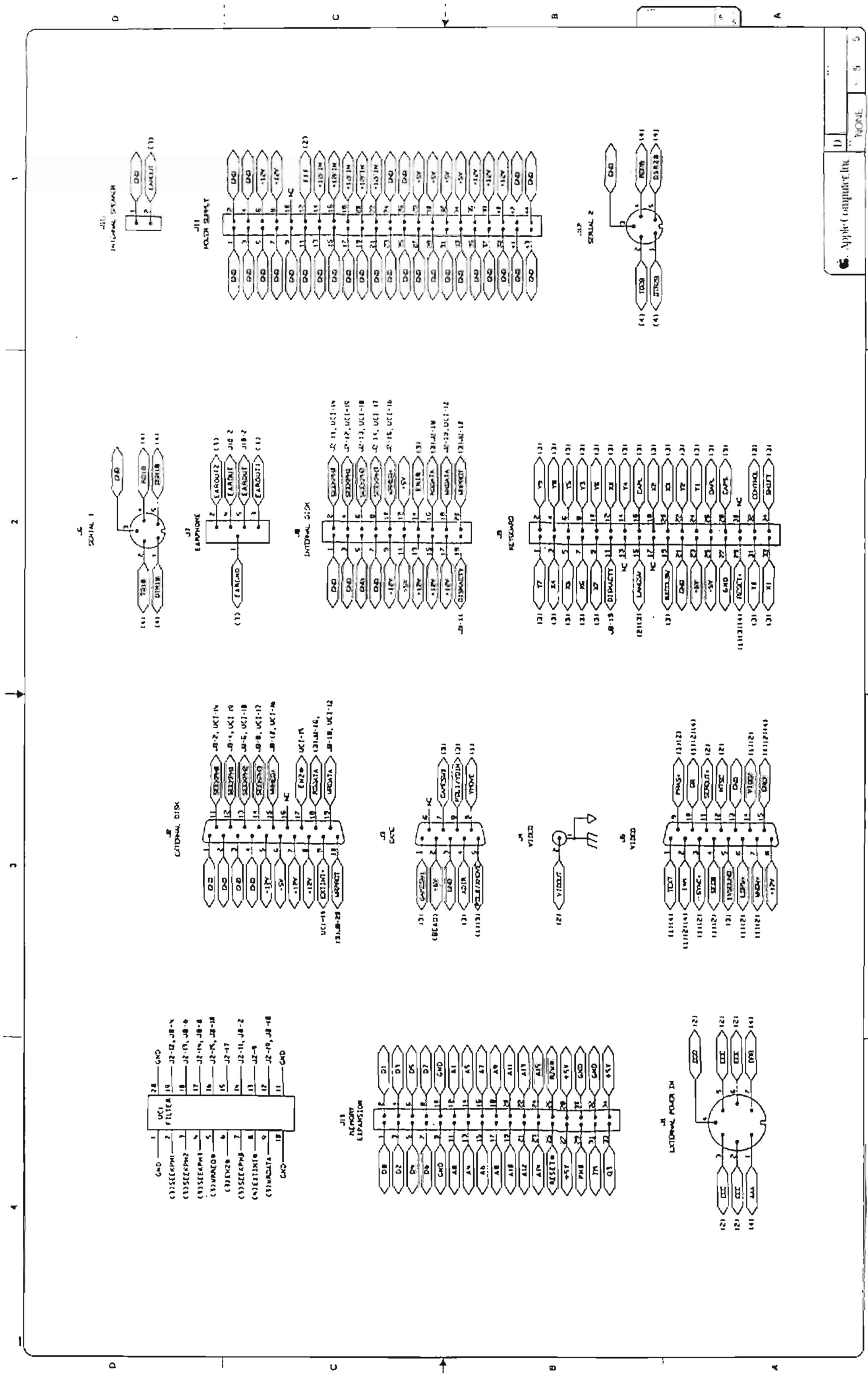
■ **Figure 11-57** Schematic diagrams for the Apple IIc (continued)



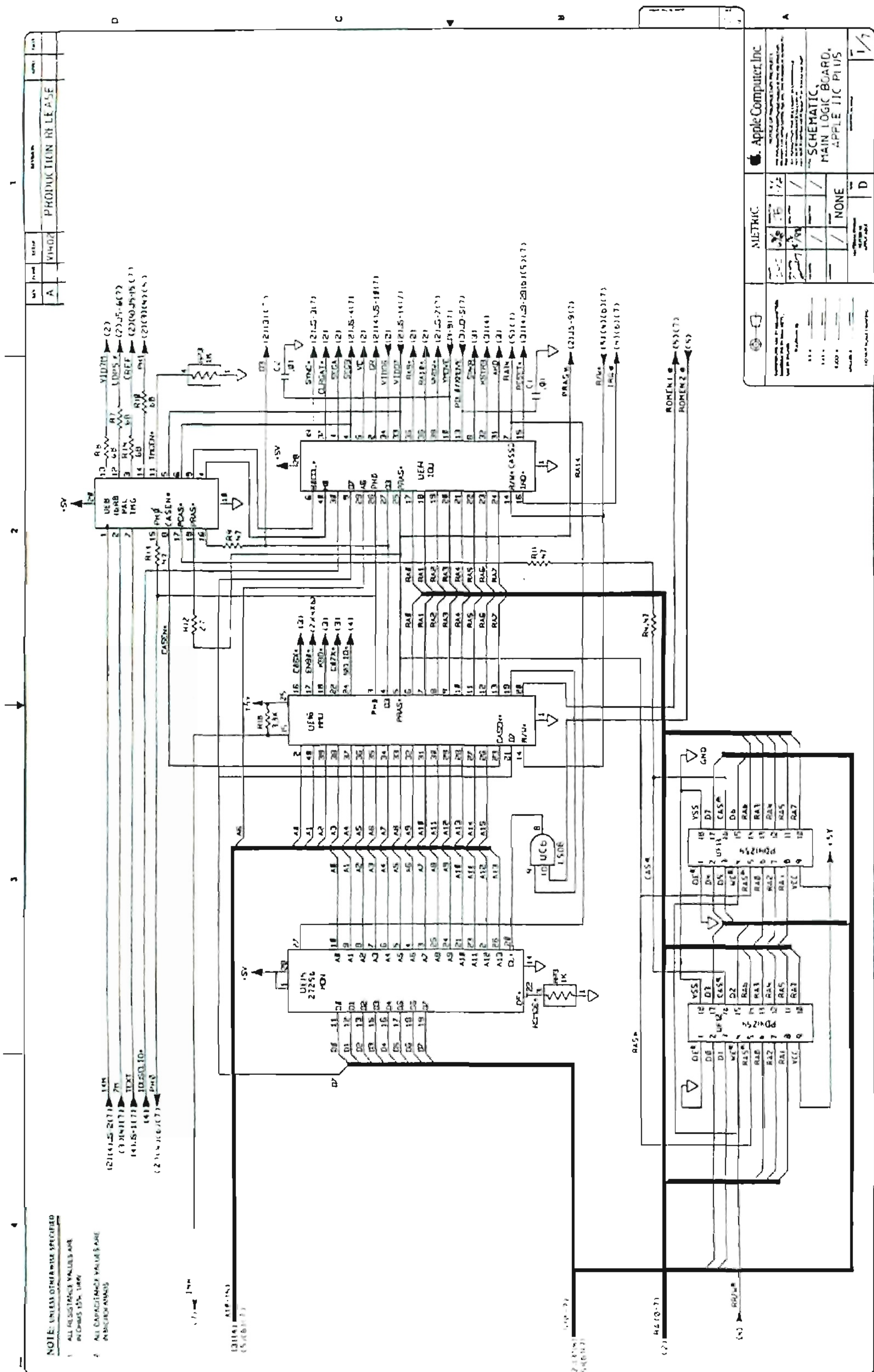
■ **Figure 11-57** Schematic diagrams for the Apple IIc (continued)



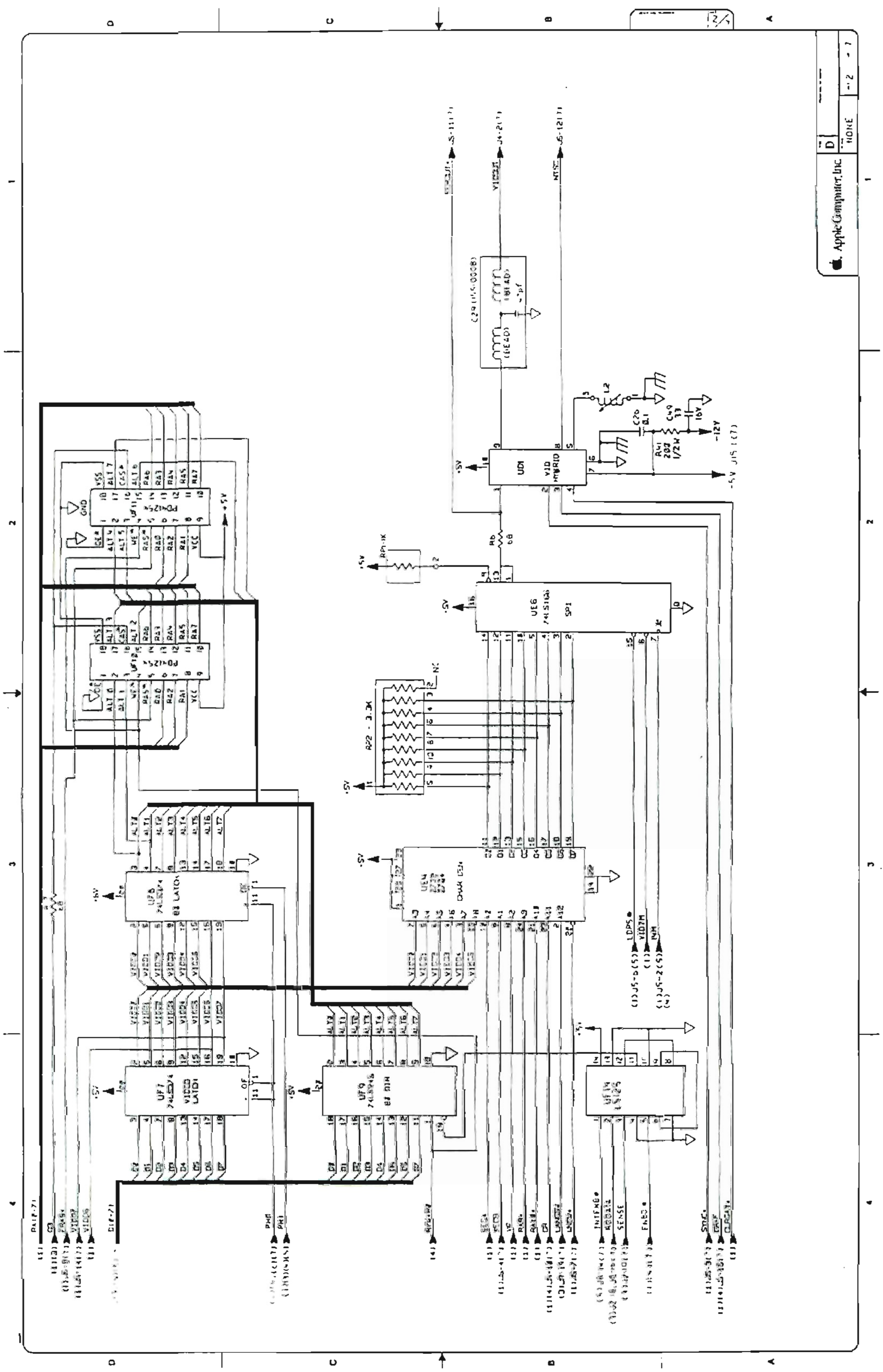
■ **Figure 11-57** Schematic diagrams for the Apple IIc (continued)



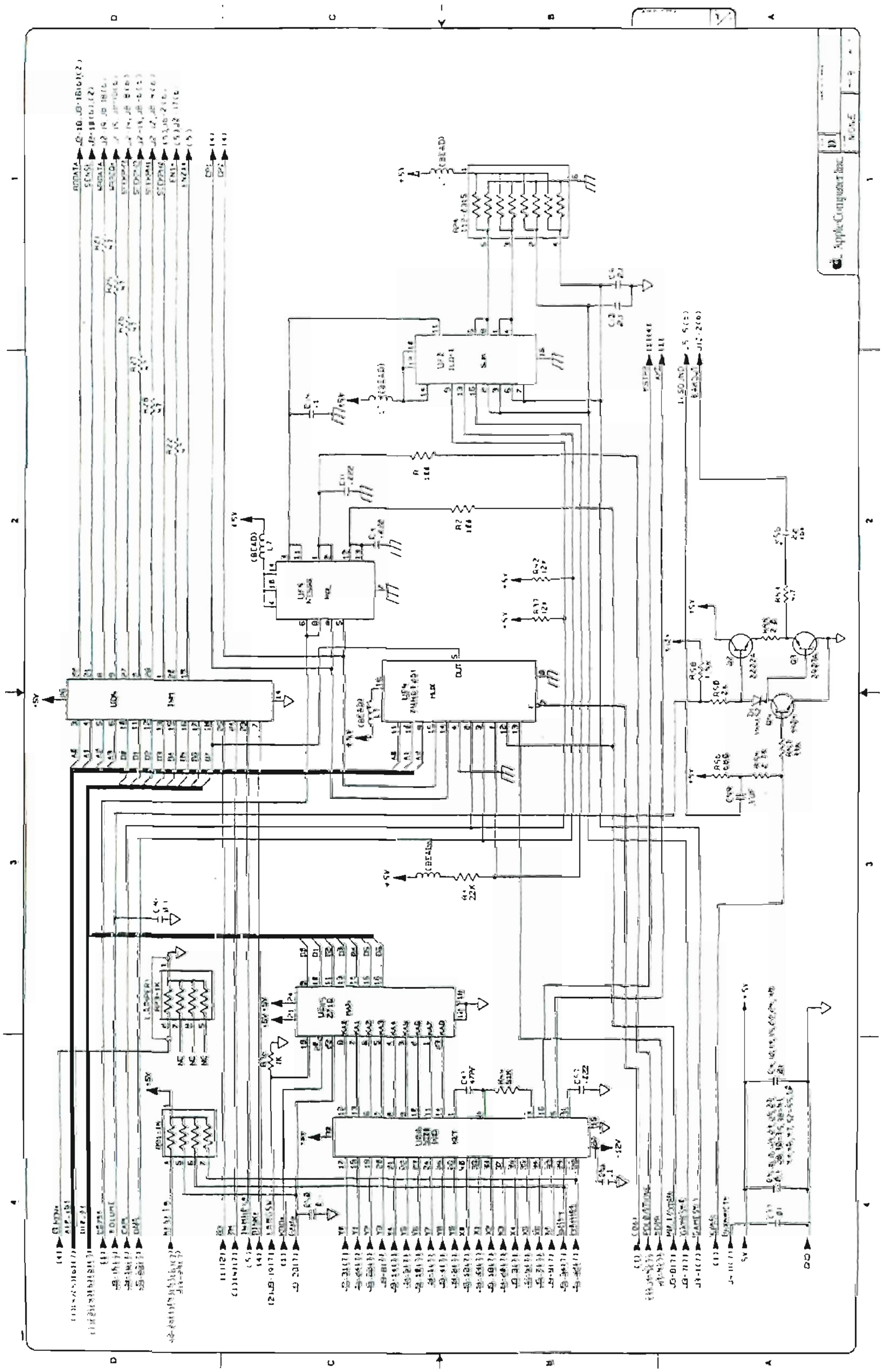
■ **Figure 11-58** Schematic diagrams for the Apple IIc Plus



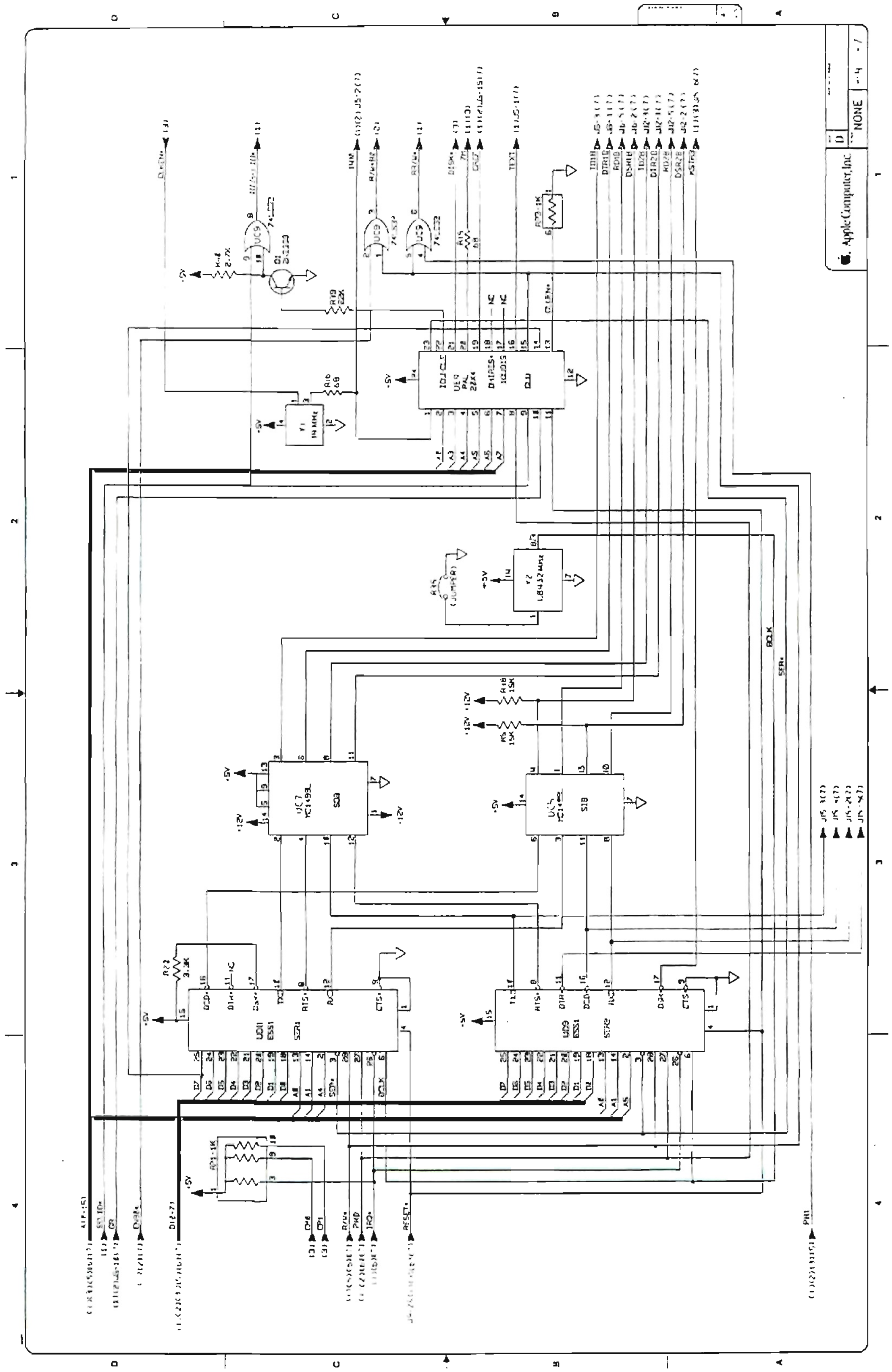
■ **Figure 11-58** Schematic diagrams for the Apple IIc Plus (continued)



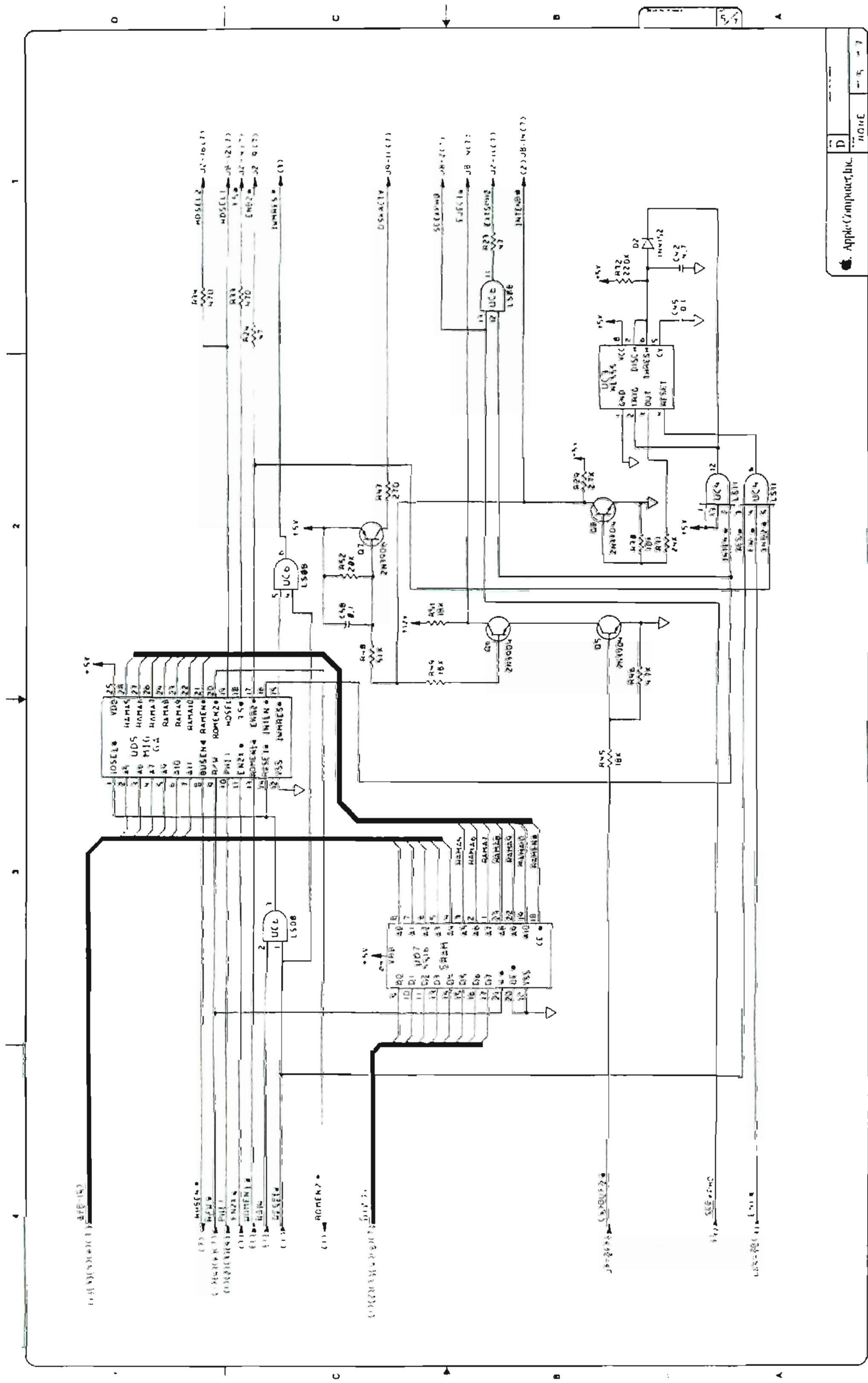
■ **Figure 11-58** Schematic diagrams for the Apple IIc Plus (continued)



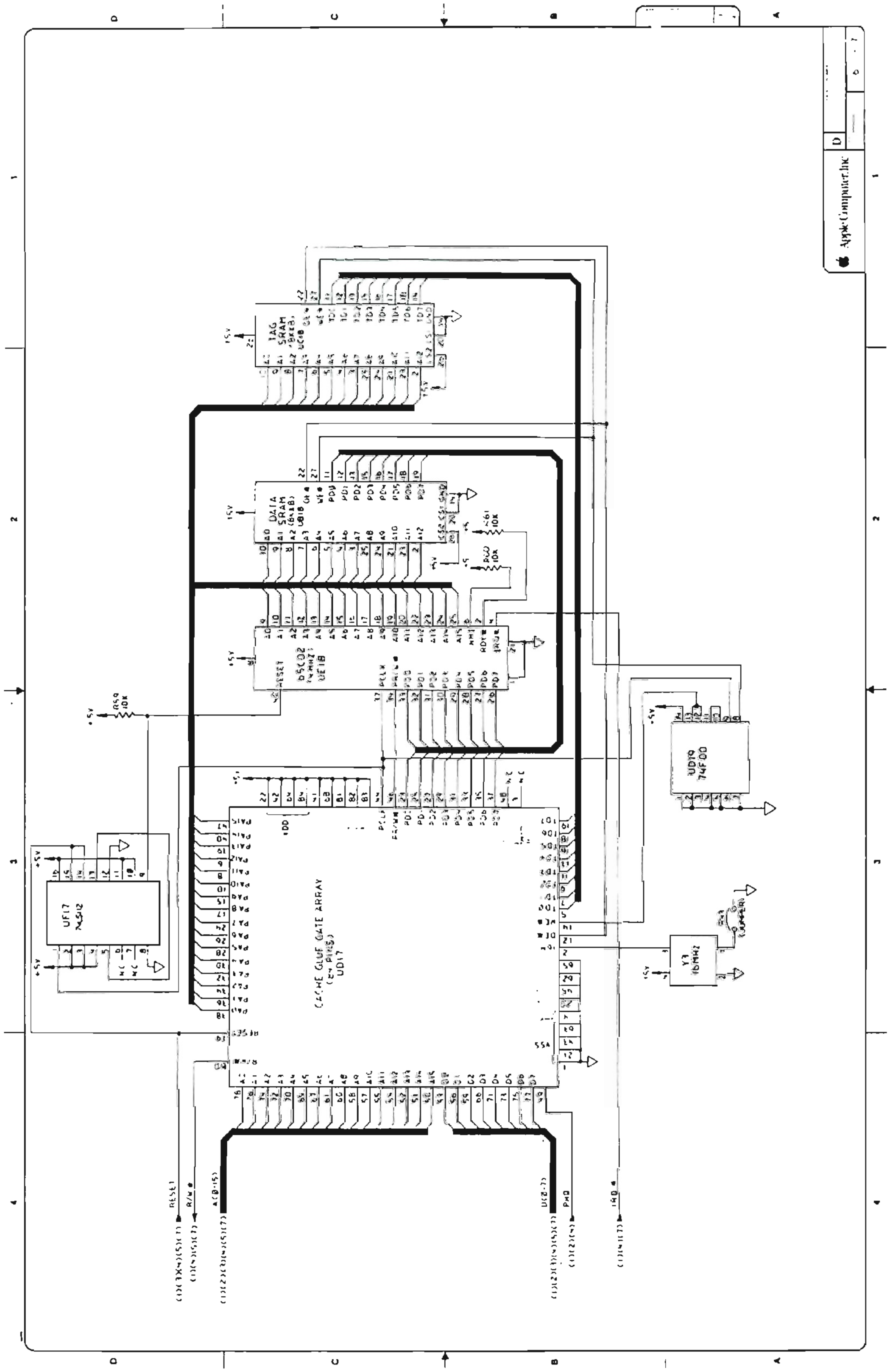
■ **Figure 11-58** Schematic diagrams for the Apple IIc Plus (continued)



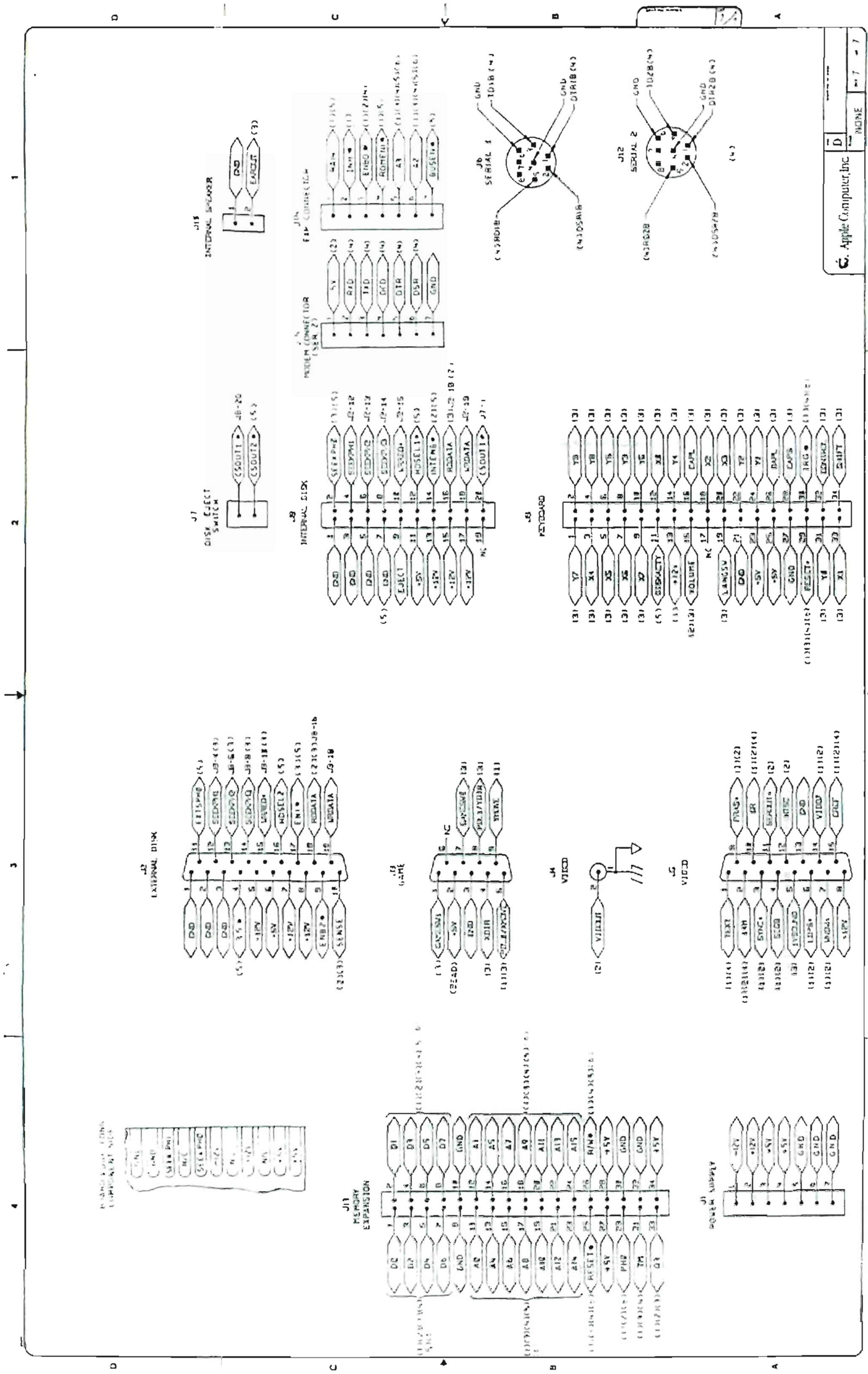
■ **Figure 11-58** Schematic diagrams for the Apple IIc Plus (continued)



■ **Figure 11-58** Schematic diagrams for the Apple IIc Plus (continued)



■ **Figure 11-58** Schematic diagrams for the Apple IIc Plus (continued)



Appendix A **The 65C02 Microprocessor**

This appendix contains the data sheet for the NCR 65C02 microprocessor. It also describes the differences between the 6502 and 65C02 microprocessors. ■

In the data sheet tables, execution times are specified in numbers of cycles. One cycle for the Apple IIc equals 0.978 microseconds. One cycle for the 65C02 in the Apple IIc Plus running at 4 MHz equals 0.25 microseconds.

If you want to write programs that execute on all computers in the Apple II series, make sure your code uses only the subset of 65C02 instructions present on the 6502.

Differences between 6502 and 65C02

The data sheet in this chapter lists the new 65C02 instructions and addressing modes. This section supplements that information by listing the instructions for which the number of cycles or the results have changed from their 6502 counterparts.

Differing cycle times

In general, differences in cycle counts are significant only in time-dependent code, such as precise wait loops. Fortunately, instructions with changed cycle counts are few. Keep in mind, however, that the 65C02 in the Apple IIc Plus computer is capable of running at 4 MHz, nearly four times as fast as the 65C02 in earlier Apple IIc computers.

Table A-1 lists the 65C02 instructions whose number of instruction execution cycles is different from their number on the 6502. See "Addressing Modes" in the data sheet for an explanation of the modes.

■ **Table A-1** Cycle time differences

Instruction/mode	Opcode	6502 cycles	65C02 cycles
ASL Absolute, X	1E	7	6
DEC Absolute, X	DE	7	6
INC Absolute, X	FE	7	6
JMP (Absolute)	6C	5	6
LSR Absolute, X	5E	7	6
ROL Absolute, X	3E	7	6
ROR Absolute, X	7E	7	6

Differing instruction results

The instructions that have different results from their 6502 equivalents are

- BIT (in immediate mode)
- JMP (indirect, when crossing a page boundary).

The BIT instruction when used in immediate mode (code $89) leaves processor status register bits 7 (N) and 6 (V) unchanged on the 65C02. On the 6502, all modes of the BIT instruction have the same effect on the status register: the value of memory bit 7 is placed in status bit 7, and memory bit 6 is placed in status bit 6. However, all BIT instructions on both versions of the processor set status bit 1 (Z) if the memory location being tested contains a 0.

If the JMP indirect instruction (code $6C) references an indirect address location that spans a page boundary, the 65C02 fetches the high-order byte of the effective address from the first byte of the next page, while the 6502 fetches it from the first byte of the current page. For example, JMP ($02FF) gets ADL from location $02FF on both processors. On the 65C02, ADH comes from $0300 while on the 6502, ADH comes from $0200.

Data sheet

The rest of this appendix is copyright 1982, NCR Corporation, Dayton, Ohio, and is reprinted with their permission.

In the data sheet, the notation for an active-low signal is an overbar instead of the asterisk used elsewhere in this manual.

NCR

NCR65C02

▪ GENERAL DESCRIPTION

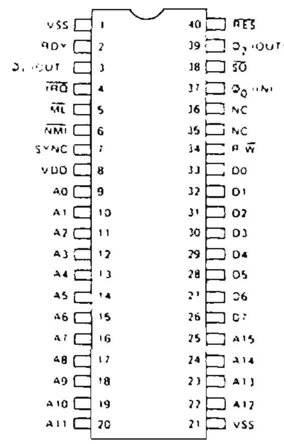
The NCR CMOS 6502 is an 8-bit microprocessor which is software compatible with the NMOS 6502. The NCR65C02 hardware interfaces with all 6500 peripherals. The enhancements include ten additional instructions, expanded operational codes and two new addressing modes. This microprocessor has all of the advantages of CMOS technology: low power consumption, increased noise immunity and higher reliability. The CMOS 6502 is a low power high performance microprocessor with applications in the consumer, business, automotive and communications market.

▪ FEATURES

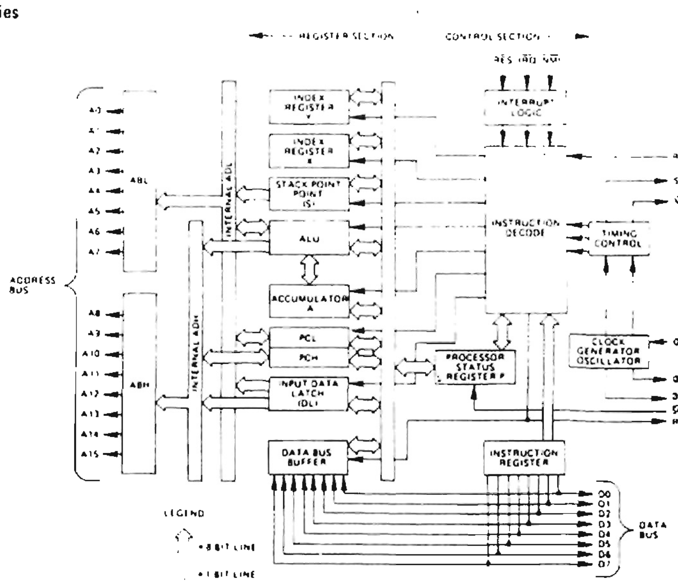
- *Enhanced software performance including 27 additional OP codes* encompassing ten new instructions and two additional addressing modes.
- 66 microprocessor instructions.
- 15 addressing modes.
- 178 operational codes.
- 1MHz, 2MHz operation.
- Operates at frequencies as low as 200 Hz for even lower power consumption (pseudo-static: stop during $\emptyset_2$ high).
- Compatible with NMOS 6500 series microprocessors.
- 64 K-byte addressable memory.
- Interrupt capability.
- Lower power consumption. 4mA @ 1MHz.
- +5 volt power supply.
- 8-bit bidirectional data bus.
- Bus Compatible with M6800.
- Non-maskable interrupt.
- 40 pin dual-in-line packaging.
- 8-bit parallel processing
- Decimal and binary arithmetic.
- Pipeline architecture.
- Programmable stack pointer.
- Variable length stack.
- Optional internal pullups for ($\overline{RDY}$, $\overline{IRQ}$, $\overline{SO}$, $\overline{NMI}$ and $\overline{RES}$)

* Specifications are subject to change without notice.

▪ PIN CONFIGURATION



▪ NCR65C02 BLOCK DIAGRAM



NCR65C02

▪ ABSOLUTE MAXIMUM RATINGS: (V_{DD} = 5.0 V ± 5%, V_{SS} = 0 V, T_A = 0° to + 70°C)

RATING	SYMBOL	VALUE	UNIT
SUPPLY VOLTAGE	V_{DD}	−0.3 to +7.0	V
INPUT VOLTAGE	V_{IN}	−0.3 to +7.0	V
OPERATING TEMP.	T_A	0 to + 70	°C
STORAGE TEMP.	T_{STG}	−55 to + 150	°C

▪ PIN FUNCTION

PIN	FUNCTION
A0 - A15	Address Bus
D0 - D7	Data Bus
$\overline{IRQ}$ *	Interrupt Request
RDY *	Ready
$\overline{ML}$	Memory Lock
$\overline{NMI}$ *	Non-Maskable Interrupt
SYNC	Synchronize
$\overline{RES}$ *	Reset
$\overline{SO}$ *	Set Overflow
NC	No Connection
R/$\overline{W}$	Read/Write
VDD	Power Supply (+5V)
VSS	Internal Logic Ground
$Ø_0$	Clock Input
$Ø_1$, $Ø_2$	Clock Output

*This pin has an optional internal pullup for a No Connect condition

▪ DC CHARACTERISTICS

	SYMBOL	MIN.	TYP.	MAX	UNIT
Input High Voltage $Ø_0$ (IN)	V_{IH}	V_{SS} + 2.4	–	V_{DD}	V
Input High Voltage $\overline{RES}$, $\overline{NMI}$, RDY, $\overline{IRQ}$, Data, S.O.		V_{SS} + 2.0	–	–	V
Input Low Voltage $Ø_0$ (IN)	V_{IL}	V_{SS} -0.3	–	V_{SS} + 0.4	V
$\overline{RES}$, $\overline{NMI}$, RDY, $\overline{IRQ}$, Data, S.O.		–	–	V_{SS} + 0.8	V
Input Leakage Current (V_{IN} = 0 to 5.25V, V_{DD} = 5.25V) With pullups	I_{IN}	-30	–	+30	μA
Without pullups		–	–	+1.0	μA
Three State (Off State) Input Current (V_{IN} = 0.4 to 2.4V, V_{CC} = 5.25V) Data Lines	I_{TSI}	–	–	10	μA
Output High Voltage (I_{OH} = −100 μAdc, V_{DD} = 4.75V SYNC, Data, A0-A15, R/W)	V_{OH}	V_{SS} + 2.4	–	–	V
Out Low Voltage (I_{OL} = 1.6mAdc, V_{DD} = 4.75V SYNC, Data, A0-A15, R/W)	V_{OL}	–	–	V_{SS} + 0.4	V
Supply Current f = 1MHz	I_{DD}	–	–	4	mA
Supply Current f = 2MHz	I_{DD}	–	–	8	mA
Capacitance (V_{IN} = 0, T_A = 25°C, f = 1MHz)	C				pF
Logic	C_{IN}	–	–	5	
Data		–	–	10	
A0-A15, R/W, SYNC	Cout	–	–	10	
$Ø_0$ (IN)	$CØ_0$ (IN)	–	–	10	

■ AC CHARACTERISTICS V_{DD} = 5.0V ± 5%, T_A = 0°C to 70°C, Load = 1 TTL + 130 pF

		1MHZ		2MHZ		3MHZ		
Parameter	Symbol	Min	Max	Min	Max	Min	Max	Unit
Delay Time, Ø0 (IN) to Ø2 (OUT)	t_{DLY}	–	60	–	60	20	60	nS
Delay Time, Ø1 (OUT) to Ø2 (OUT)	t_{DLY1}	−20	20	−20	20	−20	20	nS
Cycle Time	t_{CYC}	1.0	5000*	0.50	5000*	0.33	5000*	μS
Clock Pulse Width Low	t_{PL}	460	–	220	–	160	–	nS
Clock Pulse Width High	t_{PH}	460	–	220	–	160	–	nS
Fall Time, Rise Time	t_F, t_R	–	25	–	25	–	25	nS
Address Hold Time	t_{AH}	20	–	20	–	0	–	nS
Address Setup Time	t_{ADS}	–	225	–	140	–	110	nS
Access Time	t_{ACC}	650	–	310	–	170	–	nS
Read Data Hold Time	t_{DHR}	10	–	10	–	10	–	nS
Read Data Setup Time	t_{DSU}	100	–	60	–	60	–	nS
Write Data Delay Time	t_{MDS}	–	30	–	30	–	30	nS
Write Data Hold Time	t_{DHW}	20	–	20	–	15	–	nS
$\overline{SO}$ Setup Time	t_{SO}	100	–	100	–	100	–	nS
Processor Control Setup Time**	t_{PCS}	200	–	150	–	150	–	nS
SYNC Setup Time	t_{SYNC}	–	225	–	140	–	100	nS
$\overline{ML}$ Setup Time	t_{ML}	–	225	–	140	–	100	nS
Input Clock Rise/Fall Time	$t_{FØ0}$, $t_{RØ0}$	–	25	–	25	–	25	nS

*NCR65C02 can be held static with Ø2 high.

**This parameter must only be met to guarantee that the signal will be recognized at the current clock cycle.

■ MICROPROCESSOR OPERATIONAL ENHANCEMENTS

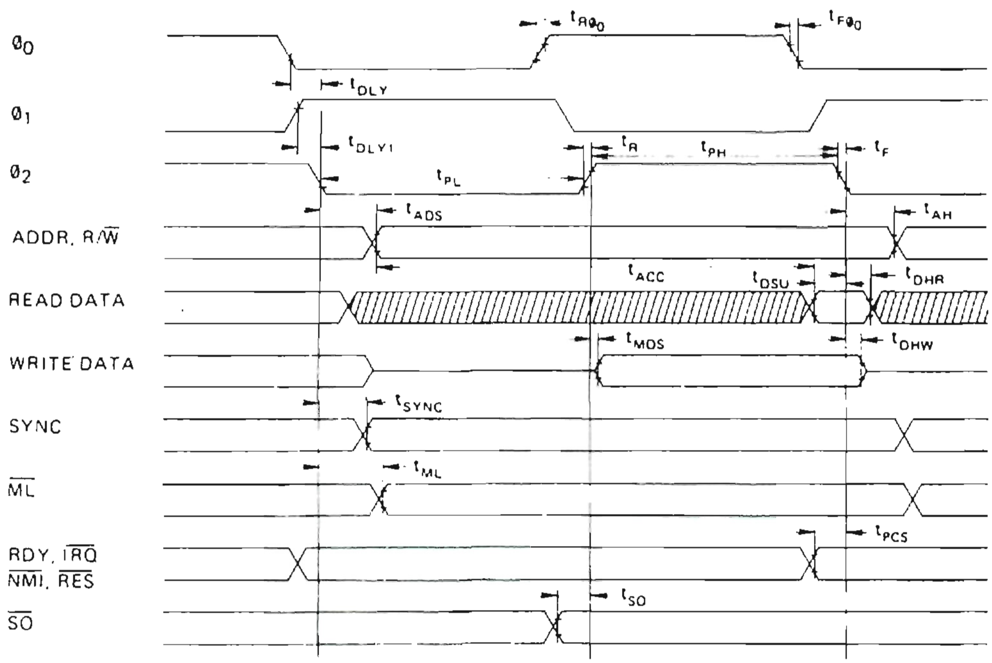
Function	NMOS 6502 Microprocessor	NCR65C02 Microprocessor
Indexed addressing across page boundary.	Extra read of invalid address.	Extra read of last instruction byte.
Execution of invalid op codes.	Some terminate only by reset. Results are undefined.	All are NOPs (reserved for future use). Op Code / Bytes / Cycles X2 / 2 / 2 X3, X7, XB, XF / 1 / 1 44 / 2 / 3 54, D4, F4 / 2 / 4 5C / 3 / 8 DC, FC / 3 / 4
Jump indirect, operand = XXFF	Page address does not increment.	Page address increments and adds one additional cycle.
Read/modify/write instructions at effective address.	One read and two write cycles.	Two read and one write cycle.
Decimal flag.	Indeterminate after reset.	Initialized to binary mode (D=0) after reset and interrupts.
Flags after decimal operation.	Invalid N, V and Z flags.	Valid flag adds one additional cycle.
Interrupt after fetch of BRK instruction.	Interrupt vector is loaded, BRK vector is ignored.	BRK is executed, then interrupt is executed.

■ MICROPROCESSOR HARDWARE ENHANCEMENTS

Function	NMOS 6502	NCR65C02
Assertion of Ready RDY during write operations.	Ignored.	Stops processor during Ø2.
Unused input-only pins ($\overline{IRQ}$, $\overline{NMI}$, RDY, $\overline{RES}$, $\overline{SO}$).	Must be connected to low impedance signal to avoid noise problems.	Connected internally by a high-resistance to V_{DD} (approximately 250 K ohm.)

NCR65C02

▪ TIMING DIAGRAM



Note. All timing is referenced from a high voltage of 2.0 volts and a low voltage of 0.8 volts.

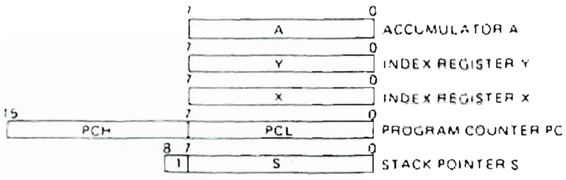
▪ NEW INSTRUCTION MNEMONICS

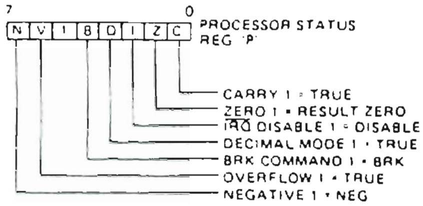
HEX	MNEMONIC	DESCRIPTION
80	BRA	Branch relative always (Relative)
3A	DEA	Decrement accumulator (Accum)
1A	INA	Increment accumulator (Accum)
DA	PHX	Push X on stack (Implied)
5A	PHY	Push Y on stack (Implied)
FA	PLX	Pull X from stack (Implied)
7A	PLY	Pull Y from stack (Implied)
9C	STZ	Store zero (Absolute)
9E	STZ	Store zero (ABS, X)
64	STZ	Store zero (Zero page)
74	STZ	Store zero (ZPG,X)
1C	TRB	Test and reset memory bits with accumulator (Absolute)
14	TRB	Test and reset memory bits with accumulator (Zero page)
0C	TSB	Test and set memory bits with accumulator (Absolute)
04	TSB	Test and set memory bits with accumulator (Zero page)

▪ ADDITIONAL INSTRUCTION ADDRESSING MODES

HEX	MNEMONIC	DESCRIPTION
72	ADC	Add memory to accumulator with carry ((ZPG))
32	AND	"AND" memory with accumulator ((ZPG))
3C	BIT	Test memory bits with accumulator (ABS, X)
34	BIT	Test memory bits with accumulator (ZPG, X)
D2	CMP	Compare memory and accumulator ((ZPG))
52	EOR	"Exclusive Or" memory with accumulator ((ZPG))
7C	JMP	Jump (New addressing mode) (ABS(IND,X))
B2	LDA	Load accumulator with memory ((ZPG))
12	ORA	"OR" memory with accumulator ((ZPG))
F2	SBC	Subtract memory from accumulator with borrow ((ZPG))
92	STA	Store accumulator in memory ((ZPG))

▪ MICROPROCESSOR PROGRAMMING MODEL





▪ FUNCTIONAL DESCRIPTION

Timing Control
The timing control unit keeps track of the instruction cycle being monitored. The unit is set to zero each time an instruction fetch is executed and is advanced at the beginning of each phase one clock pulse for as many cycles as is required to complete the instruction. Each data transfer which takes place between the registers depends upon decoding the contents of both the instruction register and the timing control unit.

Program Counter
The 16-bit program counter provides the addresses which step the microprocessor through sequential instructions in a program.

Each time the microprocessor fetches an instruction from program memory, the lower byte of the program counter (PCL) is placed on the low-order bits of the address bus and the higher byte of the program counter (PCH) is placed on the high-order 8 bits. The counter is incremented each time an instruction or data is fetched from program memory.

Instruction Register and Decode
Instructions fetched from memory are gated onto the internal data bus. These instructions are latched into the instruction register, then decoded, along with timing and interrupt signals, to generate control signals for the various registers.

Arithmetic and Logic Unit (ALU)
All arithmetic and logic operations take place in the ALU including incrementing and decrementing internal registers (except the program counter). The ALU has no internal memory and is used only to perform logical and transient numerical operations.

Accumulator
The accumulator is a general purpose 8-bit register that stores the results of most arithmetic and logic operations, and in addition, the accumulator usually contains one of the two data words used in these operations.

Index Registers
There are two 8-bit index registers (X and Y), which may be used to count program steps or to provide an index value to be used in generating an effective address.

When executing an instruction which specifies indexed addressing, the CPU fetches the op code and the base address, and modifies the address by adding the index register to it prior to performing the desired operation. Pre- or post-indexing of indirect addresses is possible (see addressing modes).

Stack Pointer
The stack pointer is an 8-bit register used to control the addressing of the variable-length stack on page one. The stack pointer is automatically incremented and decremented under control of the microprocessor to perform stack manipulations under direction of either the program or interrupts ($\overline{NMI}$ and $\overline{IRQ}$). The stack allows simple implementation of nested subroutines and multiple level interrupts. The stack pointer should be initialized before any interrupts or stack operations occur.

Processor Status Register
The 8-bit processor status register contains seven status flags. Some of the flags are controlled by the program, others may be controlled both by the program and the CPU. The 6500 instruction set contains a number of conditional branch instructions which are designed to allow testing of these flags (see microprocessor programming model).

NCR65C02

■ ADDRESSING MODES

Fifteen addressing modes are available to the user of the NCR65C02 microprocessor. The addressing modes are described in the following paragraphs.

Implied Addressing [Implied]
In the implied addressing mode, the address containing the operand is implicitly stated in the operation code of the instruction.

Accumulator Addressing [Accum]
This form of addressing is represented with a one byte instruction and implies an operation on the accumulator.

Immediate Addressing [Immediate]
With immediate addressing, the operand is contained in the second byte of the instruction, no further memory addressing is required.

Absolute Addressing [Absolute]
For absolute addressing, the second byte of the instruction specifies the eight low-order bits of the effective address, while the third byte specifies the eight high-order bits. Therefore, this addressing mode allows access to the total 64K bytes of addressable memory.

Zero Page Addressing [Zero Page]
Zero page addressing allows shorter code and execution times by only fetching the second byte of the instruction and assuming a zero high address byte. The careful use of zero page addressing can result in significant increase in code efficiency.

Absolute Indexed Addressing [ABS, X or ABS, Y]
Absolute indexed addressing is used in conjunction with X or Y index register and is referred to as "Absolute, X," and "Absolute, Y." The effective address is formed by adding the contents of X or Y to the address contained in the second and third bytes of the instruction. This mode allows the index register to contain the index or count value and the instruction to contain the base address. This type of indexing allows any location referencing and the index to modify multiple fields, resulting in reduced coding and execution time.

Zero Page Indexed Addressing [ZPG, X or ZPG, Y]
Zero page absolute addressing is used in conjunction with the index register and is referred to as "Zero Page, X" or "Zero Page, Y." The effective address is calculated by adding the second byte to the contents of the index register. Since this is a form of "Zero Page" addressing, the content of the second byte references a location in page zero. Additionally, due to the "Zero Page" addressing nature of this mode, no carry is added to the high-order eight bits of memory, and crossing of page boundaries does not occur.

Relative Addressing [Relative]
Relative addressing is used only with branch instructions; it establishes a destination for the conditional branch. The second byte of the instruction becomes the operand which is an "Offset" added to the contents of the program counter when the counter is set at the next instruction. The range of the offset is −128 to +127 bytes from the next instruction.

Zero Page Indexed Indirect Addressing [(IND, X)]
With zero page indexed indirect addressing (usually referred to as indirect X) the second byte of the instruction is added to the contents of the X index register; the carry is discarded. The result of this addition points to a memory location on page zero whose contents is the low-order eight bits of the effective address. The next memory location in page zero contains the high-order eight bits of the effective address. Both memory locations specifying the high- and low-order bytes of the effective address must be in page zero.

***Absolute Indexed Indirect Addressing [ABS(IND, X)]**
(Jump Instruction Only)
With absolute indexed indirect addressing the contents of the second and third instruction bytes are added to the X register. The result of this addition, points to a memory location containing the lower-order eight bits of the effective address. The next memory location contains the higher-order eight bits of the effective address.

Indirect Indexed Addressing [(IND), Y]
This form of addressing is usually referred to as Indirect, Y. The second byte of the instruction points to a memory location in page zero. The contents of this memory location are added to the contents of the Y index register, the result being the low-order eight bits of the effective address. The carry from this addition is added to the contents of the next page zero memory location, the result being the high-order eight bits of the effective address.

***Zero Page Indirect Addressing [(ZPG)]**
In the zero page indirect addressing mode, the second byte of the instruction points to a memory location on page zero containing the low-order byte of the effective address. The next location on page zero contains the high-order byte of the effective address.

Absolute Indirect Addressing [(ABS)]
(Jump Instruction Only)
The second byte of the instruction contains the low-order eight bits of a memory location. The high-order eight bits of that memory location is contained in the third byte of the instruction. The contents of the fully specified memory location is the low-order byte of the effective address. The next memory location contains the high-order byte of the effective address which is loaded into the 16 bit program counter.

NOTE: * = New Address Modes

■ SIGNAL DESCRIPTION

Address Bus (A0-A15)
A0-A15 forms a 16-bit address bus for memory and I/O exchanges on the data bus. The output of each address line is TTL compatible, capable of driving one standard TTL load and 130pF.

Clocks (ϕ_0, ϕ_1, and ϕ_2)
ϕ_0 is a TTL level input that is used to generate the internal clocks in the 6502. Two full level output clocks are generated by the 6502. The ϕ_2 clock output is in phase with ϕ_0. The ϕ_1 output pin is 180° out of phase with ϕ_0. (See timing diagram.)

Data Bus (D0-D7)
The data lines (D0-D7) constitute an 8-bit bidirectional data bus used for data exchanges to and from the device and peripherals. The outputs are three-state buffers capable of driving one TTL load and 130 pF.

Interrupt Request ($\overline{IRQ}$)
This TTL compatible input requests that an interrupt sequence begin within the microprocessor. The $\overline{IRQ}$ is sampled during ϕ_2 operation; if the interrupt flag in the processor status register is zero, the current instruction is completed and the interrupt sequence begins during ϕ_1. The program counter and processor status register are stored in the stack. The microprocessor will then set the interrupt mask flag high so that no further $\overline{IRQ}$s may occur. At the end of this cycle, the program counter low will be loaded from address FFFE, and program counter high from location FFFF, transferring program control to the memory vector located at these addresses. The RDY signal must be in the high state for any interrupt to be recognized. A 3K ohm external resistor should be used for proper wire OR operation.

Memory Lock ($\overline{ML}$)
In a multiprocessor system, the $\overline{ML}$ output indicates the need to defer the rearbitration of the next bus cycle to ensure the integrity of read-modify-write instructions. $\overline{ML}$ goes low during ASL, DEC, INC, LSR, ROL, ROR, TRB, TSB memory referencing instructions. This signal is low for the modify and write cycles.

Non-Maskable Interrupt ($\overline{NMI}$)
A negative-going edge on this input requests that a non-maskable interrupt sequence be generated within the microprocessor. The $\overline{NMI}$ is sampled during ϕ_2, the current instruction is completed and the interrupt sequence begins during ϕ_1. The program counter is loaded with the interrupt vector from locations FFFA (low byte) and FFFB (high byte), thereby transferring program control to the non-maskable interrupt routine.

Note: Since this interrupt is non-maskable, another $\overline{NMI}$ can occur before the first is finished. Care should be taken when using $\overline{NMI}$ to avoid this.

Ready (RDY)
This input allows the user to single-cycle the microprocessor on all cycles including write cycles. A negative transition to the low state, during or coincident with phase one (ϕ_1), will halt the microprocessor with the output address lines reflecting the current address being fetched. This condition will remain through a subsequent phase two (ϕ_2) in which the ready signal is low. This feature allows microprocessor interfacing with low-speed memory as well as direct memory access (DMA).

Reset ($\overline{RES}$)
This input is used to reset the microprocessor. Reset must be held low for at least two clock cycles after VDD reaches operating voltage from a power down. A positive transistion on this pin will then cause an initialization sequence to begin. Likewise, after the system has been operating, a low on this line of at least two cycles will cease microprocessing activity, followed by initialization after the positive edge on $\overline{RES}$.

When a positive edge is detected, there is an initialization sequence lasting six clock cycles. Then the interrupt mask flag is set, the decimal mode is cleared, and the program counter is loaded with the restart vector from locations FFFC (low byte) and FFFD (high byte). This is the start location for program control. This input should be high in normal operation.

Read/Write (R/$\overline{W}$)
This signal is normally in the high state indicating that the microprocessor is reading data from memory or I/O bus. In the low state the data bus has valid data from the microprocessor to be stored at the addressed memory location.

Set Overflow ($\overline{SO}$)
A negative transition on this line sets the overflow bit in the status code register. The signal is sampled on the trailing edge of ϕ_1.

Synchronize (SYNC)
This output line is provided to identify those cycles during which the microprocessor is doing an OP CODE fetch. The SYNC line goes high during ϕ_1 of an OP CODE fetch and stays high for the remainder of that cycle. If the RDY line is pulled low during the ϕ_1 clock pulse in which SYNC went high, the processor will stop in its current state and will remain in the state until the RDY line goes high. In this manner, the SYNC signal can be used to control RDY to cause single instruction execution.

NCR65C02

▪ INSTRUCTION SET — ALPHABETICAL SEQUENCE

ADC Add Memory to Accumulator with Carry
AND "AND" Memory with Accumulator
ASL Shift One Bit Left
BCC Branch on Carry Clear
BCS Branch on Carry Set
BEQ Branch on Result Zero
BIT Test Memory Bits with Accumulator
BMI Branch on Result Minus
BNE Branch on Result not Zero
BPL Branch on Result Plus
*BRA Branch Always
BRK Force Break
BVC Branch on Overflow Clear
BVS Branch on Overflow Set
CLC Clear Carry Flag
CLD Clear Decimal Mode
CLI Clear Interrupt Disable Bit
CLV Clear Overflow Flag
CMP Compare Memory and Accumulator
CPX Compare Memory and Index X
CPY Compare Memory and Index Y
*DEA Decrement Accumulator
DEC Decrement by One
DEX Decrement Index X by One
DEY Decrement Index Y by One
EOR "Exclusive-or" Memory with Accumulator
*INA Increment Accumulator
INC Increment by One
INX Increment Index X by One
INY Increment Index Y by One
JMP Jump to New Location
JSR Jump to New Location Saving Return Address
LDA Load Accumulator with Memory
LDX Load Index X with Memory
LDY Load Index Y with Memory
LSR Shift One Bit Right
NOP No Operation
ORA "OR" Memory with Accumulator
PHA Push Accumulator on Stack
PHP Push Processor Status on Stack
*PHX Push Index X on Stack
*PHY Push Index Y on Stack
PLA Pull Accumulator from Stack
PLP Pull Processor Status from Stack
*PLX Pull Index X from Stack
*PLY Pull Index Y from Stack
ROL Rotate One Bit Left
ROR Rotate One Bit Right
RTI Return from Interrupt
RTS Return from Subroutine
SBC Subtract Memory from Accumulator with Borrow
SEC Set Carry Flag
SED Set Decimal Mode
SEI Set Interrupt Disable Bit
STA Store Accumulator in Memory
STX Store Index X in Memory
STY Store Index Y in Memory
*STZ Store Zero in Memory
TAX Transfer Accumulator to Index X
TAY Transfer Accumulator to Index Y
*TRB Test and Reset Memory Bits with Accumulator
*TSB Test and Set Memory Bits with Accumulator
TSX Transfer Stack Pointer to Index X
TXA Transfer Index X to Accumulator
TXS Transfer Index X to Stack Pointer
TYA Transfer Index Y to Accumulator

Note: * = New Instruction

▪ MICROPROCESSOR OP CODE TABLE

S O	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	BRK	ORA ind, X			TSB* zpg	ORA zpg	ASL zpg		PHP	ORA imm	ASL A		TSB* abs	ORA abs	ASL abs		0
1	BPL rel	ORA ind, Y	ORA*† (zpg)		TRB* zpg	ORA zpg, X	ASL zpg, X		CLC	ORA abs, Y	INA* A		TRB* abs	ORA abs, X	ASL abs, X		1
2	JSR abs	AND ind, X			BIT zpg	AND zpg	ROL zpg		PLP	AND imm	ROL A		BIT abs	AND abs	ROL abs		2
3	BMI rel	AND ind, Y	AND*† (zpg)		BIT* zpg, X	AND zpg, X	ROL zpg, X		SEC	AND abs, Y	DEA* A		BIT*† abs, X	AND abs, X	ROL abs, X		3
4	RTI	EOR ind, X				EOR zpg	LSR zpg		PHA	EOR imm	LSR A		JMP abs	EOR abs	LSR abs		4
5	BVC rel	EOR ind, Y	EOR*† (zpg)			EOR zpg, X	LSR zpg, X		CLI	EOR abs, Y	PHY*			EOR abs, X	LSR abs, X		5
6	RTS	ADC ind, X			STZ* zpg	ADC zpg	ROR zpg		PLA	ADC imm	ROR A		JMP (abs)	ADC abs	ROR abs		6
7	BVS rel	ADC ind, Y	ADC*† (zpg)		STZ* zpg, X	ADC zpg, X	ROR zpg, X		SEI	ADC abs, Y	PLY*		JMP*† abs (ind, X)	ADC abs, X	ROR abs, X		7
8	BRA* rel	STA ind, X			STY zpg	STA zpg	STX zpg		DEY	BIT* imm	TXA		STY abs	STA abs	STX abs		8
9	BCC rel	STA ind, Y	STA*† (zpg)		STY zpg, X	STA zpg, X	STX zpg, Y		TYA	STA abs, Y	TXS		STZ* abs	STA abs, X	STZ* abs, X		9
A	LDY imm	LDA ind, X	LDX imm		LDY zpg	LDA zpg	LDX zpg		TAY	LDA imm	TAX		LDY abs	LDA abs	LDX abs		A
B	BCS rel	LDA ind, Y	LDA*† (zpg)		LDY zpg, X	LDA zpg, X	LDX zpg, Y		CLV	LDA abs, Y	TSX		LDY abs, X	LDA abs, X	LDX abs, Y		B
C	CPY imm	CMP ind, X			CPY zpg	CMP zpg	DEC zpg		INY	CMP imm	DEX		CPY abs	CMP abs	DEC abs		C
D	BNE rel	CMP ind, Y	CMP*† (zpg)			CMP zpg, X	DEC zpg, X		CLD	CMP abs, Y	PHX*			CMP abs, X	DEC abs, X		D
E	CPX imm	SBC ind, X			CPX zpg	SBC zpg	INC zpg		INX	SBC imm	NOP		CPX abs	SBC abs	INC abs		E
F	BEQ rel	SBC ind, Y	SBC*† (zpg)			SBC zpg, X	INC zpg, X		SED	SBC abs, Y	PLX*			SBC abs, X	INC abs, X		F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Note: * = New OP Codes
Note: † = New Address Modes

▪ OPERATIONAL CODES, EXECUTION TIME, AND MEMORY REQUIREMENTS

MNE	OPERATION		IMMEDIATE (OP n #)	ABSOLUTE (OP n #)	ZERO PAGE (OP n #)	ACCUM (OP n #)	IMPLIED (OP n #)	(IND X) (OP n #)	(IND) Y (OP n #)	ZPG X (OP n #)	ZPG Y (OP n #)	ABS X (OP n #)	ABS Y (OP n #)	RELATIVE (OP n #)	(ABS) (OP n #)	ABS (IND X) (OP n #)	(ZPG) (OP n #)	PROCESSOR STATUS CODES 7 6 5 4 3 2 1 0 / N V B D I Z C	MNE
ADC	A + M + C → A	(1 3)	69 2 2	6D 4 3	65 3 2			61 6 2	71 5 2	75 4 2		7D 4 3	79 4 3				72 5 2	N V Z C	ADC
AND	A ∧ M → A	(1)	29 2 2	2D 4 3	25 3 2			21 6 2	31 5 2	35 4 2		3D 4 3	39 4 3				32 5 2	N Z	AND
ASL	C ← 7 0 ← 0	(1)		0E 6 3	06 5 2	0A 2 1				16 6 2		1E 6 3						N Z C	ASL
BCC	Branch if C=0	(2)												90 2 2					BCC
BCS	Branch if C=1	(2)												B0 2 2					BCS
BEQ	Branch if Z=1	(2)												F0 2 2					BEQ
BIT	A ∧ M	(4 5)	89 2 2	2C 4 3	24 3 2					34 4 2		3C 4 3						M_7 M_6 Z	BIT
BMI	Branch if N=1	(2)												30 2 2					BMI
BNE	Branch if Z=0	(2)												D0 2 2					BNE
BPL	Branch if N=0	(2)												10 2 2					BPL
BRA	Branch Always	(2)												80 2 2					BRA
BRK	Break						00 7 1											1 1	BRK
BVC	Branch if V=0	(2)												50 2 2					BVC
BVS	Branch if V=1	(2)												70 2 2				0	BVS
CLC	0 → C						18 2 1											0	CLC
CLD	0 → D						D8 2 1											0	CLD
CLI	0 → I						58 2 1											0	CLI
CLV	0 → V						B8 2 1											0	CLV
CMP	A − M	(1)	C9 2 2	CD 4 3	C5 3 2			C1 6 2	D1 5 2	D5 4 2		DD 4 3	D9 4 3				D2 5 2	N Z C	CMP
CPX	X − M		E0 2 2	EC 4 3	E4 3 2													N Z C	CPX
CPY	Y − M		C0 2 2	CC 4 3	C4 3 2													N Z C	CPY
DEA	A − 1 → A					3A 2 1												N Z	DEA
DEC	M − 1 → M	(1)		CE 6 3	C6 5 2					D6 6 2		DE 6 3						N Z	DEC
DEX	X − 1 → X						CA 2 1											N Z	DEX
DEY	Y − 1 → Y						88 2 1											N Z	DEY
EOR	A ⊻ M → A	(1)	49 2 2	4D 4 3	45 3 2			41 6 2	51 5 2	55 4 2		5D 4 3	59 4 3				52 5 2	N Z	EOR
INA	A + 1 → A					1A 2 1												N Z	INA
INC	M + 1 → M	(1)		EE 6 3	E6 5 2					F6 6 2		FE 6 3						N Z	INC
INX	X + 1 → X						E8 2 1											N Z	INX
INY	Y + 1 → Y						C8 2 1											N Z	INY
JMP	Jump to new loc			4C 3 3											6C 6 3	7C 6 3			JMP
JSR	Jump Subroutine			20 6 3															JSR
LDA	M → A	(1)	A9 2 2	AD 4 3	A5 3 2			A1 6 2	B1 5 2	B5 4 2		BD 4 3	B9 4 3				B2 5 2	N Z	LDA
LDX	M → X	(1)	A2 2 2	AE 4 3	A6 3 2						B6 4 2		BE 4 3					N Z	LDX
LDY	M → Y	(1)	A0 2 2	AC 4 3	A4 3 2					B4 4 2		BC 4 3						N Z	LDY
LSR	0 → 7 0 → C	(1)		4E 6 3	46 5 2	4A 2 1				56 6 2		5E 6 3						0 Z C	LSR
NOP	PC + 1 → PC						EA 2 1												NOP
ORA	A V M → A	(1)	09 2 2	0D 4 3	05 3 2			01 6 2	11 5 2	15 4 2		1D 4 3	19 4 3				12 5 2	N Z	ORA
PHA	A → M_S S − 1 → S						48 3 1												PHA
PHP	P → M_S S − 1 → S						08 3 1												PHP
PHX	X → M_S S − 1 → S						DA 3 1												PHX
PHY	Y → M_S S − 1 → S						5A 3 1												PHY
PLA	S + 1 → S M_S → A						68 4 1											N Z	PLA
PLP	S + 1 → S M_S → P						28 4 1											N V 1 0 1 Z C	PLP
PLX	S + 1 → S M_S → X						FA 4 1											N Z	PLX
PLY	S + 1 → S M_S → Y						7A 4 1											N Z	PLY
ROL		(1)		2E 6 3	26 5 2	2A 2 1				36 6 2		3E 6 3						N Z C	ROL
ROR		(1)		6E 6 3	66 5 2	6A 2 1				76 6 2		7E 6 3						N Z C	ROR
RTI	Return from Inter						40 6 1											N V 1 0 1 Z C	RTI
RTS	Return from Subr						60 6 1												RTS
SBC	A − M − $\bar{C}$ → A	(1 3)	E9 2 2	ED 4 3	E5 3 2			E1 6 2	F1 5 2	F5 4 2		FD 4 3	F9 4 3				F2 5 2	N V Z C	SBC
SEC	1 → C						38 2 1											1	SEC
SED	1 → D						F8 2 1											1	SED
SEI	1 → I						78 2 1											1	SEI
STA	A → M			8D 4 3	85 3 2			81 6 2	91 6 2	95 4 2		9D 5 3	99 5 3				92 5 2		STA
STX	X → M			8E 4 3	86 3 2						96 4 2								STX
STY	Y → M			8C 4 3	84 3 2					94 4 2									STY
STZ	00 → M			9C 4 3	64 3 2					74 4 2		9E 5 3							STZ
TAX	A → X						AA 2 1											N Z	TAX
TAY	A → Y						A8 2 1											N Z	TAY
TRB	$\bar{A}$ ∧ M → M	(4)		1C 6 3	14 5 2													Z	TRB
TSB	A V M → M	(4)		0C 6 3	04 5 2													Z	TSB
TSX	S → X						BA 2 1											N Z	TSX
TXA	X → A						8A 2 1											N Z	TXA
TXS	X → S						9A 2 1												TXS
TYA	Y → A						98 2 1											N Z	TYA

Notes

1. Add 1 to "n" if page boundary is crossed
2. Add 1 to "n" if branch occurs to same page
   Add 2 to "n" if branch occurs to different page
3. Add 1 to "n" if decimal mode
4. V bit equals memory bit 6 prior to execution
   N bit equals memory bit 7 prior to execution

*5. The immediate addressing mode of the BIT instruction leaves bits 6 & 7 (V & N) in the Processor Status Code Register unchanged.

X Index X
Y Index Y
A Accumulator
M Memory per effective address
Ms Memory per stack pointer

+ Add
− Subtract
∧ And
V Or
⊻ Exclusive or

n No Cycles
No Bytes
M_6 Memory bit 6
M_7 Memory bit 7

Appendix B **Address Map**

This appendix lists all important RAM and hardware locations in address order and briefly describes them.

The tables in this appendix list addresses in either two or three forms: the hexadecimal form (preceded by a dollar sign) for use in assembly language; the decimal form for use in Applesoft BASIC; and (for numbers greater than 32,767) the complementary decimal value for use in Apple Integer BASIC. ■

Page $00

Table B-1 lists the zero-page addresses in hexadecimal and decimal form, followed by symbols denoting the firmware or system software that uses them.

- *M* denotes the Monitor.
- *A* denotes Applesoft BASIC.
- *I* denotes Integer BASIC.
- *D* denotes DOS 3.3.
- *P* denotes ProDOS. Locations whose contents ProDOS saves and restores have a P in parentheses, indicating that ProDOS has no net effect on them.

■ **Table B-1** Page \$00 use

Hex	Dec	Used by	Hex	Dec	Used by
\$00	0	A	\$25	37	M
\$01	1	A	\$26	38	M D
\$02	2	A	\$27	39	M D
\$03	3	A	\$28	40	M D
\$04	4	A	\$29	41	M D
\$05	5	A	\$2A	42	M D
\$06	6		\$2B	43	M D
\$07	7		\$2C	44	M D
\$08	8		\$2D	45	M D
\$09	9		\$2E	46	M D
\$0A	10	A	\$2F	47	M D
\$0B	11	A	\$30	48	M
\$0C	12	A	\$31	49	M
\$0D	13	A	\$32	50	M
\$0E	14	A	\$33	51	M
\$0F	15	A	\$34	52	M
\$10	16	A	\$35	53	M D
\$11	17	A	\$36	54	M D
\$12	18	A	\$37	55	M D
\$13	19	A	\$38	56	M D
\$14	20	A	\$39	57	M D
\$15	21	A	\$3A	58	M P
\$16	22	A	\$3B	59	M P
\$17	23	A	\$3C	60	M P
\$18	24	A	\$3D	61	M P
\$19	25		\$3E	62	M D P
\$1A	26		\$3F	63	M D P
\$1B	27		\$40	64	M D (P)
\$1C	28		\$41	65	M D(P)
\$1D	29		\$42	66	M D (P)
\$1F	31		\$43	67	M D (P)

■ **Table B-1** Page $00 use (continued)

Hex	Dec	Used by					Hex	Dec	Used by		
$44	68	M			D	(P)	$67	103	A	I	D
$45	69	M			D	(P)	$68	104	A	I	D
$46	70	M			D	ß(P)	$69	105	A	I	D
$47	71	M			D	(P)	$6A	106	A	I	D
$48	72	M			D	(P)	$6B	107	A	I	
$49	73	M				(P)	$6C	108	A	I	
$4A	74			I	D	(P)	$6D	109	A	I	
$4B	75			I	D	(P)	$6E	110	A	I	
$4C	76			I	D	(P)	$6F	111	A	I	D
$4D	77			I	D	(P)	$70	112	A	I	D
$4F	79	M					$71	113	A	I	
$55	85	M	A	I			$72	114	A	I	
$56	86		A	I			$73	115	A	I	
$57	87		A	I			$74	116	A	I	
$58	88		A	I			$75	117	A	I	
$59	89		A	I			$76	118	A	I	
$5A	90		A	I			$77	119	A	I	
$5B	91		A	I			$78	120	A	I	
$5C	92		A	I			$79	121	A	I	
$5D	93		A	I			$7A	122	A	I	
$5E	94		A	I			$7B	123	A	I	
$5F	95		A	I			$7C	124	A	I	
$60	96		A	I			$7D	125	A	I	
$61	97		A	I			$7E	126	A	I	
$62	98		A	I			$7F	127	A	I	
$63	99		A	I			$80	128	A	I	
$64	100		A	I			$81	129	A	I	
$65	101		A	I			$82	130	A	I	
$66	102		A	I			$83	131	A	I	

■ **Table B-1** Page $00 use (continued)

Hex	Dec	Used by			Hex	Dec	Used by		
$84	132	A	I		$A2	162	A	I	
$85	133	A	I		$A3	163	A	I	
$86	134	A	I		$A4	164	A	I	
$87	135	A	I		$A5	165	A	I	
$88	136	A	I		$A6	166	A	I	
$89	137	A	I		$A7	167	A	I	
$8A	138	A	I		$A8	168	A	I	
$8B	139	A	I		$A9	169	A	I	
$8C	140	A	I		$AA	170	A	I	
$8D	141	A	I		$AB	171	A	I	
$8E	142	A	I		$AC	172	A	I	
$8F	143	A	I		$AD	173	A	I	
$90	144	A	I		$AE	174	A	I	
$91	145	A	I		$AF	175	A	I	D
$92	146	A	I		$B0	176	A	I	D
$93	147	A	I		$B1	177	A	I	
$94	148	A	I		$B2	178	A	I	
$95	149	A	I		$B3	179	A	I	
$96	150	A	I		$B4	180	A	I	
$97	151	A	I		$B5	181	A	I	
$98	152	A	I		$B6	182	A	I	
$99	153	A	I		$B7	183	A	I	
$9A	154	A	I		$B8	184	A	I	
$9B	155	A	I		$B9	185	A	I	
$9C	156	A	I		$BA	186	A	I	
$9D	157	A	I		$BB	187	A	I	
$9E	158	A	I		$BC	188	A	I	
$9F	159	A	I		$BD	189	A	I	
$A0	160	A	I		$BE	190	A	I	
$A1	161	A	I		$BF	191	A	I	

■ **Table B-1** Page $00 use (continued)

Hex	Dec	Used by			Hex	Dec	Used by
$C0	192	A	I		$E0	224	A
$C1	193	A	I		$E1	225	A
$C2	194	A	I		$E2	226	A
$C3	195	A	I		$E3	227	
$C4	196	A	I		$E4	228	A
$C5	197	A	I		$E5	229	A
$C6	198	A	I		$E6	230	A
$C7	199	A	I		$E7	231	A
$C8	200	A	I		$E8	232	A
$C9	201	A	I		$E9	233	A
$CA	202	A	I	D	$EA	234	A
$CB	203	A	I	D	$EB	235	
$CC	204	A	I	D	$EC	236	
$CD	205	A	I	D	$ED	237	
$CE	206		I		$EE	238	
$CF	207		I		$EF	239	
$D0	208	A	I		$F0	240	A
$D1	209	A	I		$F1	241	A
$D2	210	A	I		$F2	242	A
$D3	211	A	I		$F3	243	A
$D4	212	A	I		$F4	244	A
$D5	213	A	I		$F5	245	A
$D6	214		I		$F6	246	A
$D7	215		I		$F7	247	A
$D8	216	A	I	D	$F8	248	A
$D9	217	A	I		$F9	249	
$DA	218	A	I		$FA	250	
$DB	219	A	I		$FB	251	
$DC	220	A	I		$FC	252	
$DD	221	A	I		$FD	253	
$DE	222	A	I		$FE	254	
$DF	223	A	I		$FF	255	

Page $03

Most of page $03 is available for small machine-language programs. The built-in Monitor uses the top 16 addresses of page $03, as shown in Table B-2; the XFer routine uses locations $03ED and $03EE. If you are using DOS or ProDOS, it also uses the 32 locations $03D0 through $03EF.

■ **Table B-2** Page $03 use

Hex	Dec	Use
$03F0	1008	Address of the subroutine that handles BRK instructions (normally $FA59)
$03F1	1009	
$03F2	1010	Reset vector
$03F3	1011	
$03F4	1012	Power-up byte (see text)
$03F5	1013	Jump instruction to Applesoft &-command
$03F6	1014	handler (initially $4C, $58, $FF)
$03F7	1015	
$03F8	1016	Jump instruction to user Control-Y command
$03F9	1017	handler
$03FA	1018	
$03FB	1019	Jump instruction to NMI interrupt handler
$03FC	1020	(not used by Apple IIc)
$03FD	1021	
$03FE	1022	Address of user IRQ interrupt handler
$03FF	1023	

Screen holes

One result of the way the Apple IIc–family hardware maps display memory on the screen is that groups of 8 memory addresses are left over in 16 areas of the text and Lo-Res graphics display pages—8 areas in main RAM and 8 in auxiliary RAM. The firmware uses for these 128 bytes are shown in Tables B-3 and B-4.

△ Apple IIc Plus Memory expansion — The memory expansion Apple IIc and the Apple IIc Plus use some of the screen holes differently than earlier Apple IIc computers. Where they differ, the memory expansion/Apple IIc Plus ROM assignments are given in braces ({ }) following the original and UniDisk 3.5 assignments. △

■ **Table B-3** Main memory screen hole allocations

Hex	Dec	Description
$0478	1144	Mouse port: low byte of clamping minimum
$0479	1145	Reserved for serial port 1
$047A	1146	Reserved for serial port 2
$047B	1147	Reserved
$047C	1148	Low byte of X coordinate {Reserved}
$047D	1149	Reserved for mouse port
$047E	1150	Reserved
$047F	1151	Reserved {Low byte of X coordinate}
$04F8	1272	Mouse port: low byte of clamping maximum
$04F9	1273	Reserved for serial port 1
$04FA	1274	Reserved for serial port 2
$04FB	1275	Reserved
$04FC	1276	Low byte of Y coordinate {Owner of serial buffer (if any)}
$04FD	1277	Reserved for mouse port
$04FE	1278	Reserved
$04FF	1279	Owner of serial buffer (if any) {Low byte of Y coordinate}

■ **Table B-3** Main memory screen hole allocations (continued)

Hex	Dec	Description
$04FD	1277	Reserved for mouse port
$04FE	1278	Reserved
$04FF	1279	Owner of serial buffer (if any) {Low byte of Y coordinate}
$0578	1400	Mouse port: high byte of clamping minimum
$0579	1401	Port 1 printer width (1–255; 0 = unlimited)
$057A	1402	Port 2 line length (1–255; 0 = unlimited)
$057B	1403	Cursor horizontal position (80-column display)
$057C	1404	High byte of X coordinate {Pointer to next storage location for keyboard buffer}
$057D	1405	Reserved for mouse port
$057E	1406	Reserved
$057F	1407	Pointer to next storage location for serial buffer {High byte of X coordinate}
$05F8	1528	Mouse port: high byte of clamping maximum
$05F9	1529	Port 1 temporary storage location
$05FA	1530	Keyboard buffer control
$05FB	1531	Reserved
$05FC	1532	High byte of Y coordinate {Pointer to next storage location for keyboard buffer}
$05FD	1533	Reserved for mouse port
$05FE	1534	Reserved
$05FF	1535	Pointer to next storage location for keyboard buffer {High byte of Y coordinate}
$0678	1656	Reserved
$0679	1657	Indicates when port 1 firmware is parsing a command
$067A	1658	Indicates when port 2 firmware is parsing a command
$067B	1659	Reserved
$067C	1660	Mouse port: reserved {Pointer to next retrieval location for serial buffer}
$067D	1661	Reserved for mouse port
$067E	1662	Reserved
$067F	1663	Pointer to next retrieval location for serial buffer {Mouse port: reserved}
$06F8	1784	Reserved
$06F9	1785	Current port 1 command character
$06FA	1786	Current port 2 command character

■ **Table B-3** Main memory screen hole allocations (continued)

Hex	Dec	Description
$06FB	1787	Reserved
$06FC	1788	Mouse port: reserved (Pointer to next retrieval location for keyboard buffer)
$06FD	1789	Reserved for mouse port
$06FE	1790	Reserved
$06FF	1791	Pointer to next retrieval location for keyboard buffer (Mouse port: reserved)
$0778	1912	$n0 = current active port number x 16
$0779	1913	Port 1 flags for echo and auto line feed
$077A	1914	Port 2 flags for each and auto line feed
$077B	1915	Reserved
$077C	1916	Mouse port status byte (Reserved)
$077D	1917	Reserved for mouse port
$077E	1918	Reserved
$077F	1919	Reserved (Mouse port status byte)
$07F8	2040	Owner of $C800–$CFFF ($C3, video)
$07F9	2041	Port 1 current printer column
$07FA	2042	Port 2 current line position
$07FB	2043	Reserved
$07FC	2044	Mouse port mode byte (Reserved)
$07FD	2045	Reserved for mouse port
$07FE	2046	Reserved
$07FF	2047	Reserved (Mouse port mode byte)

■ **Table B-4** Auxiliary memory screen hole allocations

Hex	Dec	Description
$0478	1144	Initial port 1 ACIA control register values ($9E)
$0479	1145	Initial port 1 ACIA command register values ($0B)
$047A	1146	Initial port 1 characteristics flags ($40)
$047B	1147	Initial port 1 printer width ($50)
$047C	1148	Initial port 2 ACIA control register values ($16)
$047D	1149	Initial port 2 ACIA command register values ($0B)
$047E	1150	Initial port 2 characteristics flags ($01)
$047F	1151	Initial port 2 line length ($00)
$04F8 through $04FF	1272 1279	Reserved
$0578 through $057F	1400 1407	Reserved
$05F8 through $05FF	1528 1535	Reserved
$0678 through $067F	1656 1663	Reserved
$06F8 through $06FF	1784 1791	Reserved
$0778 through $077F	1912 1919	Reserved
$07F8 through $07FF	2040 2047	Reserved

The hardware page ($C0)

Tables B-5 through B-9 list all the hardware locations available for use in the Apple IIc family. These tables have a column at the left that is not present in other tables in this appendix. This column, labeled *RW*, indicates the action to take at a particular location.

- *R* means read.
- *RR* means read twice in succession.
- *R7* means read the byte and then check bit 7; in the use column, "See if..." refers to the condition represented by bit 7 = 1, unless otherwise specified. Bit 7 has a value of $80, so if the contents of the location are greater than or equal to $80, the bit is on.

  Another way to test bit 7 (the sign bit) is with a BIT instruction, followed by BPL (bit 7 was 0) or BMI (bit 7 was 1).
- *R/W* means to either read or write. For writing, the value is unimportant.
- *W* means to write only. The value is unimportant.
- *N* means not to read or write because the location is reserved.

An address of the form $C00n refers to the 16 locations from $C000 through $C00F. Labels, when they are shown, are simply memory aids. Some of them correspond to the labels at those addresses in the firmware, while others do not. Your program must assign a label for it anyway.

■ **Table B-5** Addresses $C000–$C03F

RW	Hex	Dec	Neg dec	Label	Use
R	$C00n			KStrb	Read keyboard data (bits 0–6) and strobe (bit 7)
W	$C000	49152	–16384	80Store	Off: Page2 switches Page 1 and 2
W	$C001	49153	–16383	80Store	On: Page2 switches Page 1 and 1X
W	$C002	49154	–16382	RAMRd	Off: Read main 48K memory
W	$C004	49156	–16380	RAMWrt	Off: Write in main 48K memory
W	$C005	49157	–16379	RAMWrt	On: Write in auxiliary 48K memory
W	$C006	49158	–16378		Reserved
W	$C007	49159	–16377		Reserved
W	$C008	49160	–16376	AltZP	Off: Use main P0, P1, bank-switched RAM

■ **Table B-5** Addresses $C000–$C03F (continued)

RW	Hex	Dec	Neg dec	Label	Use
W	$C009	49161	–16375	AltZP	On: Use auxiliary P0, P1, bank-switched RAM
W	$C00A	49162	–16374		Reserved
W	$C00B	49163	–16373		Reserved
W	$C00C	49164	–16372	80Col	Off: 40-column display
W	$C00D	49165	–16371	80Col	On: 80-column display
W	$C00E	49166	–16270	AltChar	Off: Display primary character set
W	$C00F	49167	–16369	AltChar	On: Display alternate character set
W	$C01n				Clear keyboard strobe ($C00n bit 7)
R7	$C010	49168	–16368	AKD	See if any key now down; clear strobe
R7	$C011	49169	–16367	RdBnk2	See if using $D000 bank 2 (or 1)
R7	$C012	49170	–16366	RdLCRAM	See if reading RAM (or ROM).
R7	$C013	49171	–16365	RdRAMRd	See if reading auxiliary or main 48K memory
R7	$C014	49172	–16364	RdRAMWrt	See if writing auxiliary or main 48K memory
R	$C015	49173	–16363	RstXInt	Reset mouse X0 interrupt
R7	$C016	49174	–16362	RdAltZP	See if auxiliary P0, P1 and bank-switched RAM
R	$C017	49175	–16361	RstYInt	Reset mouse Y interrupt
R7	$C018	49176	–16360	Rd80Store	See if 80Store on (or off)
R7	$C019	49177	–16359	RstVBl	Read and then reset VBL interrupt flag
R7	$C01A	49178	–16358	RdTEXT	See if text (or graphics)
R7	$C01B	49179	–16357	RdMIXED	See if mixed mode switch on
R7	$C01C	49180	–16356	RdPage2	See if Page 2/1X selected
R7	$C01D	49181	–16355	RdHiRes	See if Hi-Res switch on
R7	$C01E	49182	–16354	RdAltChar	See if alternate character set (or primary)
R7	$C01F	49183	–16353	Rd80Col	Read 80Col switch (1=on)
R/W	$C020	49184	–16352		
	through				Toggle between main and auxiliary ROM
R/W	$C02F	49199	–1633		
W	$C030	49200	–16336		Reserved
R	$C030	49200	–16336		Toggle speaker
N	$C031	49201	–16335		
	through				Reserved (read and write)
N	$C03F	49215	–16321		

■ **Table B-6** Addresses $C040–$C05F

RW	Hex	Dec	Neg dec	Label	Use
R7	$C040	49216	-16320	RdXYMsk	See if X0/Y0 mask set
R7	$C041	49217	-16319	RdVBlMsk	See if VBL mask set
R7	$C042	49218	-16318	RdX0Edge	See if interrupt on falling X0 edge
R7	$C043	49219	-16317	RdY0Edge	See if interrupt on falling Y0 edge
N	$C044	49220	-16316		Reserved
N	$C045	49221	-16315)		Reserved
N	$C046	49222	-16314		Reserved
N	$C047	49223	-16313		Reserved
R	$C048	49224	-16312	RstXY	Reset X0/Y0 interrupt flags
N	$C049	49225	-16311		Reserved
N	$C04A	49226	-16310		Reserved
N	$C04B	49227	-16309		Reserved
N	$C04C	49228	-16308		Reserved
N	$C04D	49229	-16307		Reserved
N	$C04E	49230	-16306		Reserved
N	$C04F	49231	-16305		Reserved
R/W	$C050	49232	-16304	Text	Off: Graphics display
R/W	$C051	49233	-16303	Text	On: Text display
R/W	$C052	49234	-16302	Mixed	Off: Text or graphics only
R/W	$C053	49235	-16301	Mixed	On: Combination text and graphics
R/W	$C054	49236	-16300	Page2	Off: Use Page 1
R/W	$C055	49237	-16299	Page2	On: Display Page 2 (80Store off); store to Page 1X (80Store on)
R/W	$C056	49238	-16298	HiRes	Off: Lo-Res
R/W	$C057	49239	-16297	HiRes	On: Hi-Res; Double Hi-Res If 80Col and DHiRes on
N	$C058	49240	-16296		Reserved if IOUDis on ($C07E bit 7=1)
R/W				DisX	Disable (mask) mouse X0/Y0 interrupts
N	$C059	49241	-16295		Reserved if IOUDis on
R/W				EnbXY	Enable (allow) mouse X0/Y0 interrupts
N	$C05A	49242	-16294		Reserved if IOUDis on
R/W				DisVBl	Disable (mask) VBL interrupts
N	$C05B	49243	-16293		Reserved if IOUDis on
R/W				EnVBl	Enable (allow) VBL interrupts
N	$C05C	49244	-16292		Reserved if IOUDis on
R/W				X0Edge	Interrupt on rising edge of X0

■ **Table B-6** Addresses \$C040–\$C05F (continued)

RW	Hex	Dec	Neg dec	Label	Use
N	\$C05D	49245	–1629		Reserved if IOUDis on
R/W				X0Edge	Interrupt on falling edge of X0
R/W	\$C05E	49246	–16290	DHiRes	If IOUDis on: Set Double Hi-Res
R/W				Y0Edge	If IOUDis off: Interrupt on rising Y0
R/W	\$C05F	49247	–16289	DHiRes	If IOUDis on: Clear Double Hi-Res
R/W				Y0Edge	If IOUDis off: Interrupt on falling Y0

■ **Table B-7** Addresses \$C060–\$C07F

RW	Hex	Dec	Neg dec	Label	Use
W	\$C06x				Reserved (write)
R7	\$C060	4924	–16288	Rd80Sw	See if 80/40 switch down (1 = 40)
R7	\$C061	49249	–16287	RdBtn0	See if mouse button/paddle button 0/ Command (Open-Apple) key pressed (1 = pressed)
R7	\$C062	49250	–16286	RdBtn1	See if paddle button 1/Option (Solid Apple) pressed (1 = pressed)
R7	\$C063	49251	–16285	Rd63	See if mouse button not pressed (0 = pressed)
R7	\$C064	49252	–16284	Pdl0	Paddle 0 analog input
R7	\$C065	49253	–16283	Pdl1	Paddle 1 analog input
R7	\$C066	49254	–16282	MouX1	See if mouse X1 (direction) is high
R7	\$C067	49255	–16281	MouY1	See if mouse Y1 (direction) is high
N	\$C068	49256	–16280		
	through				Reserved (write and read)
N	\$C06F	49263	–16273		
R/W	\$C07x				Trigger paddle timer; reset VBlInt; however, \$C071–\$C07D are reserved
R/W	\$C070	49264	–16272	PTrig	Designated trigger or reset location
N	\$C071	49265	–16271		
	through				Reserved
N	\$C07D	49277	–16259		

■ **Table B-7** Addresses $C060–$C07F

RW	Hex	Dec	Neg dec	Label	Use
R7	$C07E	49278	–16258	RdIOUDis	See if IOUDis on; trigger paddle timer; reset VBlInt
W				IOUDis	On: Enable access to DHiRes switch; disable $C058–$C05F IOU access
R7	$C07F	49279	–16257	RdDHiRes	See if DHiRes off
W				IOUDis	Off: Disable access to DHiRes switch; enable $C058–$C05F IOU access

■ **Table B-8** Addresses $C080–$C0AF

RW	Hex	Dec	Neg dec	Label	Use
R	$C080	49280	–16256		Read RAM; no write; use $D000 bank 2
RR	$C081	49281	–16255		Read ROM; write RAM; use $D000 bank 2
R	$C082	49282	–16254		Read ROM; no write; use $D000 bank 2
RR	$C083	49283	–16253		Read and write RAM; use $D000 bank 2
N	$C084	49284	–16252		Reserved
N	$C085	49285	–16251		Reserved
N	$C086	49286	–16250		Reserved
N	$C087	49287	–16249		Reserved
R	$C088	49288	–16248		Read RAM; no write; use $D000 bank 1
RR	$C089	49289	–16247		Read ROM; write RAM; use $D000 bank 1
R	$C08A	49290	–16246		Read ROM; no write; use $D000 bank 1
RR	$C08B	49291	–16245		Read and write RAM; use $D000 bank 1
N	$C08C	49292	–16244		Reserved
N	$C08D	49293	–16243		Reserved
N	$C08E	49294	–16242		Reserved
N	$C08F	49295	–16241		Reserved
N	$C090	49296	–16240		
	through				Reserved
N	$C097	49303	–16233		
R/W	$C098	49304	–16232		Port 1 ACIA transmit/receive register
R/W	$C099	49305	–16231		Port 1 ACIA status register
R/W	$C09A	49306	–16230		Port 1 ACIA command register
R/W	$C09B	49307	–16229		Port 1 ACIA control register

■ **Table B-8** Addresses $C080–$C0AF (continued)

RW	Hex	Dec	Neg dec	Label	Use
N	$C09C	49308	–16228		
	through				Reserved
N	$C09F	49311	–16225		
N	$C0A0	49312	–16224		
	through				Reserved
N	$C0A7	49319	–16217		
R/W	$C0A8	49320	–16216		Port 2 ACIA transmit/receive register
R/W	$C0A9	49321	–16215		Port 2 ACIA status register
R/W	$C0AA	49322	–16214		Port 2 ACIA command register
R/W	$C0AB	49323	–16213		Port 2 ACIA control register
N	$C0AC	49324	–16212		
	through				Reserved
N	$C0AF	49327	–16209		

■ **Table B-9** Addresses $C0B0–$C0FF

RW	Hex	Dec	Neg Dec	Label	Use
N	$C0B0	49328	–16208		
	through				Reserved
N	$C0BF	49343	–16193		
N	$C0C0	49344	–16192		
	through				Reserved
N	$C0CF	49359	–16177		
N	$C0D0	49360	–16176		
	through				Reserved
N	$C0DF	49375	–16161		
N	$C0E0	49376	–16160		
	through				Reserved
N	$C0EF	49391	–16145		
N	$C0F0	49392	–16144		
	through				Reserved
N	$C0FF	49407	–16129		

Appendix C **Operating Systems and Languages**

This appendix is an overview of the characteristics of operating systems and languages when run on Apple IIc-family computers. It is not intended to be a complete description. For more information, refer to the manuals that are provided with each product. ■

Operating systems

This section discusses the operating systems with which the Apple IIc family works. CP/M—and any other operating system that requires an interface card—does not work on the Apple IIc family.

ProDOS

ProDOS is the preferred disk operating system for the Apple IIc family. It supports all of the hardware and firmware features of all versions of the Apple IIc.

DOS

The Apple IIc family works with DOS 3.3. Its built-in disk drive hardware and firmware can also access DOS 3.2 disks by using the *BASICS* disk. DOS support is provided for the sake of Apple II-series compatibility; neither version of DOS takes full advantage of all the features of the Apple IIc family.

Pascal Operating System

Versions 1.2 and later of the Pascal Operating System use the 80/40 switch and the interrupt features of the Apple IIc family, while remaining compatible with the other Apple II-series computers. Note that the Apple IIc Plus does not have an 80/40 switch.

While the Apple IIc family works with Pascal 1.1, this version of the Pascal Operating System does not use the 80/40 switch or handle interrupts.

The Apple IIc family does not work with Pascal 1.0, because the I/O firmware entry points of that version of the operating system are rigidly defined (rather than being accessed via a table), and the Apple IIc family operating firmware does not correspond to these entry points.

Languages

This section discusses using Apple programming languages with the Apple IIc family.

Applesoft BASIC

The programming examples in this manual are almost entirely in assembly language, and so most addresses and values are given in hexadecimal notation.

Use a PEEK in BASIC (instead of LDA in assembly language) to read a location, and a POKE (instead of STA) to write to a location. The values used by Applesoft must be in decimal, so you will have to convert hexadecimal values given in this manual to decimal. (Several tables in this manual include decimal equivalents to make the job easier for you.)

If you read a hardware address from a BASIC program, you get a value between 0 and 255. Bit 7 has a value of 128, so if a soft switch is on, its value will be equal to or greater than 128; if the switch is off, the value will be less than 128.

Integer BASIC

You will have to run a version of DOS in your Apple IIc family computer to use Integer BASIC. ProDOS does not support Integer BASIC.

Pascal language

The Pascal language runs on the Apple IIc family under versions 1.1 or later of the Pascal Operating System. However, for best performance, use Pascal versions 1.2 or later.

Fortran

Fortran runs under version 1.1 of the Pascal Operating System, which does not detect or use certain Apple IIc family features, such as the 80/40 switch or auxiliary memory. Therefore, Fortran does not take advantage of these features either.

Logo II

Apple Logo II works under ProDOS on Apple II-series machines with at least 128 KB of memory. Logo II is a version of the Logo language originally developed from the LISP (*LIS*t *P*rocessing) language at MIT as a language to be used for learning. Logo II takes advantage of Apple II graphics and retains much of the power and flavor of LISP without LISP's somewhat cryptic syntax.

Appendix D **Apple II–Family Differences**

This appendix compares the Apple IIc to the Apple IIc Plus, Apple IIe, Apple II Plus, and Apple II. It does not contain an exhaustive list of differences, but it does mention those differences most likely to affect the accuracy of programs, displays, and instructions created for end users of two or more Apple II-series models.

The Apple IIGS computer differs in many important ways from all other Apple II computers. The Apple IIGS is not discussed in this appendix; see the *Technical Introduction to the Apple IIGS* for a summary of the features of the Apple IIGS computer. ■

Overview

The differences between the computers in the Apple II series can be expressed as equations: this computer equals that one plus or minus certain features.

The following equations compare each model of Apple II series with its predecessor in terms of functional equivalence, not literal equality. For example,

Apple II Plus = Apple II – Integer BASIC firmware

does not mean that Integer BASIC firmware can be removed from the Apple II—just that the one machine functions as if it were the other without such firmware.

Apple II Plus	=	**Apple II**	+	Autostart ROM
			+	Applesoft firmware
			+	48 KB RAM standard
			–	old Monitor ROM
			–	Integer BASIC firmware
Apple IIe	=	**Apple II Plus**	+	Apple Language Card (with 16 KB of RAM)
			+	80-column (enhanced) video firmware
			+	built-in diagnostics
			+	full ASCII keyboard
			+	internal power light
			+	FCC approval
			+	improved back panel
			+	9-pin back-panel game connector
			+	auxiliary slot (with possibility of 80-column text card and extra 64 KB RAM)
			–	slot 0
			+	interrupt support in firmware (enhanced Apple IIe)
			+	mini-assembler in firmware (enhanced Apple IIe)
			+	18-key numeric keypad (extended keyboard Apple IIe)

Apple IIc	=	Apple IIe	+	extended 80-column text card
			+	80/40 switch
			+	keyboard switch
			+	disk-use light
			+	disk controller port
			+	built-in 5.25-inch disk drive
			+	mouse port
			+	serial printer port
			+	serial communication port
			+	built-in port firmware
			+	video expansion connector
			+	external power transformer
			–	removable cover
			–	slots 1 to 7
			–	auxiliary slot
			–	internal power light
			–	cassette I/O connectors
			–	internal game I/O connector
			–	auxiliary video pin
			–	Monitor cassette support
			+	mini-assembler in firmware (UniDisk 3.5 and later versions)
			+	SmartPort in firmware (UniDisk 3.5 and later versions)
			+	memory expansion card support (memory expansion version)
Apple IIc Plus	=	Apple IIc	+	4 MHz 65C02 microprocessor
			+	CGGA and cache RAM
			+	SmartPort in firmware
			+	mini-assembler in firmware
			+	memory expansion card support
			+	connector for internal modem
			+	Apple Standard Keyboard layout
			+	all-internal power supply
			+	mini-DIN connectors for serial ports
			+	support for security kit
			+	firmware driver for Apple 3.5 disk drives
			+	volume control on keyboard
			+	built-in 3.5-inch disk drive with eject button
			–	built-in 5.25-inch disk drive
			–	1 MHz 65C02 microprocessor
			–	80/40 switch
			–	volume control knob on side of case
			–	external audio jack

Type of processor

The processor in the Apple II and II Plus is the 6502. The original Apple IIe uses a 6502A. The Apple IIc family and later versions of the Apple IIe all use the 65C02; a redesigned CMOS 6502 with 27 new instructions, new addressing modes, and, for some instructions, a differing execution scheme and machine cycle counts (see Appendix A). The Apple IIc Plus uses a faster (4 MHz) version of the 65C02.

A program will run on earlier Apple II machines only if it does not contain instructions unique to the 65C02 or depend on the timing of any of those instructions whose cycle times differ between the two processors.

Your program is guranteed *not* to run on all versions of the Apple II if it uses any firmware entry points other than those in Appendix F.

Machine identification

Your Apple II program can tell which member of the Apple II family it is running on by checking the values in four locations of ROM (the *identification bytes*). Table D-1 lists the machines and their identification bytes.

■ **Table D-1** Apple II–family identification bytes

	Location			
Machine	**$FBB3**	**$FB1E**	**$FBC0**	**$FBBF**
Apple II	$38		$60	$2F
Apple II Plus	$EA	$AD	$EA	$EA
Apple III (emulation mode)	$EA	$8A		
Apple IIe (original)	$06		$EA	$C1
Apple IIe (enhanced)	$06		$E0	$00
Apple IIc (original)	$06		$00	$FF
Apple IIc (UniDisk 3.5)	$06		$00	$00
Apple IIc (memory expansion)	$06		$00	$03
Apple IIc (memory expansion, revised ROM)	$06		$00	$04
Apple IIc Plus	$06		$00	$05

Any future Apple II–series computer or ROM release will have different values in these locations. Machine identification routines are available from Apple Developer Technical Support.

The MachID byte for ProDOS ($BF98 on the global page) has bit 3 set to 0 if the computer is an Apple II, II Plus, IIe, or III, and to 1 if the computer is not one of these machines. In an Apple IIc family member, bits 7 and 6 are also set to binary 10. Bits 7 and 6 set to binary 10 indicate that a computer is Apple IIe and IIc compatible, regardless of the value of bit 3.

Memory structure

This section compares the memory organization of the Apple IIc and the Apple IIc Plus with that of the Apple II, II Plus, and IIe. These machines differ in RAM space, ROM space, slot or port address space, and hardware page use.

Amount and address ranges of RAM

The Apple II could have as little as 4 KB of RAM at the time of purchase, and could be upgraded to as much as 48 KB of RAM.

The Apple II Plus has 48 KB of RAM ($0000 through $BFFF) as a standard feature. With the addition of an Apple Language Card, a 48 KB Apple II or II Plus could be expanded to have 64 KB of RAM.

The Apple IIe has a full 64 KB of RAM. The top 12K addresses overlap with the ROM addresses $D000 through $FFFF. There is an additional bank-switched area of 4 KB from $D000 through $DFFF. This arrangement is equivalent to an Apple II Plus with an Apple Language Card installed. A program selects between the RAM and ROM address spaces and between the $D000 address spaces by changing soft switches located in memory.

With an Apple IIe Extended 80-Column Text Card installed in its auxiliary slot, an Apple IIe has an additional 64 KB of RAM available, although no more than half of the 128 KB of RAM space is available at any given time. Soft switches located in memory control these address space selections. The Extended 80-Column Text Card is included with each enhanced Apple IIe, and comes permanently installed in the extended keyboard Apple IIe.

The RAM in the Apple IIc and Apple IIc Plus is equivalent to the RAM in an Apple IIe with an Extended 80-Column Text Card. The optional memory expansion card can add as much as 1 MB of RAM to the memory expansion Apple IIc or Apple IIc Plus in 256 KB steps.

Amount and address ranges of ROM

The Apple II has 8 KB of ROM ($E000 through $FFFF), and the Apple II Plus has 12 KB of ROM ($D000 through $FFFF). Users can plug their own ROMs into the sockets provided. The on-board (as opposed to slot) ROM address range is from $D000 through $FFFF.

The Apple IIe has 16 KB of ROM, of which it uses 15.75 KB (addresses $C100 through $FFFF; page $C0 addresses are for I/O hardware). ROM addresses $C300 through $C3FF (normally assigned to the ROM in a card in slot 3) and $C800 through $CFFF contain 80-column video firmware; ROM addresses $C100 through $C2FF and $C400 through $C7FF (normally assigned to the ROM on cards in slots 1, 2, 4, 5, 6, and 7) contain built-in self-test routines.

A soft switch in RAM controls whether the video firmware or slot 3 card ROM is active. Invoking the self-tests with Option-Control-Reset causes the self-test firmware to take over the slot ROM address spaces.

The Apple IIc–family ROM also uses the 15.75 KB from $C100 through $FFFF, and its enhanced video firmware has the same entry point addresses as on the Apple IIe. However, there are only rudimentary built-in self-tests, and these do not preempt any port firmware space.

The UniDisk 3.5 and memory expansion versions of the Apple IIc and the Apple IIc Plus have twice the ROM (32 KB) of the original Apple IIc. The extra ROM contains support for the SmartPort, a mini-assembler, Step and Trace functions in the Monitor firmware, expanded self-test routines, improved interrupt support, and, in the Apple IIc Plus, a disk driver for Apple 3.5 disks. Whereas most of the second 16 KB of ROM in the Apple IIc is empty, in the Apple IIc Plus, all of the available ROM is used.

In the Apple IIc and Apple IIc Plus, main ROM addresses $C100 through $CFFF contain I/O and interrupt firmware, addresses $D000 through $F7FF contain the Applesoft BASIC interpreter, and addresses $F800 through $FFFF contain the Monitor.

Peripheral-card memory spaces

Each port in an Apple IIc–family computer has up to 16 peripheral-card I/O space locations in main memory on the hardware page (beginning at location $C0n0 + $80 for slot or port n), allocated in the standard Apple II–series way (that is, beginning at location $C0n0 + $80 for each slot n).

The peripheral-card ROM space (page $Cn for slot n in the Apple II, II Plus, and IIe) contains the starting and entry-point addresses for port n, but port routines are not limited to their allocated $Cn pages.

The 2 KB expansion ROM space from $C800 to $CFFF in the Apple IIc family is used by the enhanced video firmware and miscellaneous I/O and memory-transfer routines.

The 128 bytes of peripheral-card RAM space or scratch-pad RAM (64 screen holes in main memory and their equivalent addresses in auxiliary memory) are reserved for use by the built-in firmware. It is extremely important for the correct operation of Apple IIc–family firmware that these locations not be altered by software except for the specific purposes described in Chapters 3, 7, and 9.

Hardware addresses

The hardware page (the addresses from $C000 through $C0FF) controls memory selection and input/output hardware characteristics. All input and output (except video output) takes place at one or more hardware page addresses. For the sake of simplicity, this section presents only a general comparison between the Apple IIc family on the one hand, and the Apple II, II Plus, and IIe on the other, with respect to hardware page use. However, for many characteristics, the Apple IIe and IIc work one way, while the Apple II and II Plus work another.

$C000–$C00F

On all Apple II–series computers, reading any one of these addresses reads the keyboard data and strobe. On the Apple IIe and IIc, writing to each of these addresses turns memory and display switches on and off. Writing to addresses $C006, $C007, $C00A, and $C00B performs ROM selection on the Apple IIe. Writing to these four addresses is reserved in the Apple IIc family.

For reading the keyboard, use $C000; reserve $C001 through $C00F.

$C010–$C01F

On all Apple II–series computers, writing to any one of these addresses clears the keyboard strobe. On the Apple IIe and IIc, reading each of these addresses checks the status of a memory or display switch, or the any-key-down flag.

For clearing the keyboard strobe, use $C010; reserve $C011 through $C01F.

Reading $C015 checks the SLOTCXROM switch on the Apple IIe, but it resets the X-movement interrupt (XInt) in the Apple IIc family. Similarly, reading $C017 checks the SLOTC3ROM switch on the Apple IIe, but it resets the Y-movement interrupt (YInt) in the Apple IIc family.

Reading $C019 checks the current state of vertical blanking (VBL) on the Apple IIc, but it reads and then resets the vertical blanking interrupt flag in the Apple IIc family.

$C020–$C02F

On the Apple II, II Plus, and IIe, reading any address $C02x toggles the cassette output signal. On the original Apple IIc, both reading from and writing to these locations are reserved. Apple IIc versions with 32 KB of ROM—including the Apple IIc Plus—use $C028 to switch in or out the extra 16 KB of ROM.

$C030–$C03F

On all Apple II–series computers, reading an address of the form $C03x toggles the speaker. For full Apple II–series compatibility, toggle the speaker using $C030, and reserve $C031 through $C03F.

For the Apple IIc family, writing to $C031 through $C03F is explicitly reserved.

$C040–$C04F

On the Apple II, II Plus, and IIe, reading any address of the form $C04x triggers the utility strobe. The Apple IIc family has no utility strobe.

For the Apple IIc family, addresses $C044 through $C047 are explicitly reserved, and reading or writing any address from $C048 through $C04F resets both the X and Y mouse interrupts (XInt and YInt).

$C050–$C05F

Addresses $C050 through $C057 work the same on the Apple IIc family as on the Apple IIe: they turn the Text, Mixed, Page2, and HiRes switches on and off.

On the Apple IIe, addresses $C058 through $C05F turn the annunciator outputs on and off. On an Apple IIe with a revision B main logic board or later, an Apple IIe Extended 80-Column Text Card, and a jumper installed on the card, reading locations $C05E and $C05F set and clear Double Hi-Res display mode.

In the Apple IIc family, if the IOUDis switch is on, both reading from and writing to addresses $C058 through $C05D are reserved, and addresses $C05E and $C05F set and clear the Double Hi-Res display (as on the Apple IIe equipped as described in the preceding paragraph). If the IOUDis switch is off, then addresses $C058 through $C05F control various characteristics of mouse and vertical blanking interrupts (listed in Table 9-2).

$C060–$C06F

For the Apple IIc family, writing to any address of the form $C06x is reserved, and reading addresses $C068 through $C06F is reserved.

Reading addresses $C061 and $C062 is the same as on the Apple IIe (switch inputs, Command, and Option). Reading addresses $C064 and $C065 is the same as on all other Apple II–series computers (analog inputs 0 and 1).

For the Apple IIc family, address $C063 bit 7 is 1 if the mouse switch is not pressed, and 0 if it is pressed, so that software looking for the Shift-key mod (used on Apple II, II Plus, and IIe with some text cards) will find it and display lowercase correctly. If by chance the mouse button is pressed when the software checks location $C063, it will appear that the Shift-key mod is not present.

On the Apple IIc, address $C060 is used for reading the state of the 80/40 switch; on the Apple II, II Plus, and IIe, this address is for reading cassette input. On the Apple IIc Plus, this address is reserved.

The Apple IIc family has two, rather than four, analog (paddle) inputs. Addresses $C066 and $C067 are used for reading the mouse X and Y direction bits.

$C070–$C07F

On the Apple II, II Plus, and IIe, reading from or writing to any address of the form $C07n triggers the paddle timers (analog input).

In the Apple IIc family, only address $C070 is to be used for that one function. Addresses $C071 through $C07D are explicitly reserved. The results of reading from or writing to addresses $C07E and $C07F are described in Table 6-9.

$C080–$C08F

On the Apple IIe and IIc, accessing addresses in this range selects different combinations of bank-switched memory banks. However, addresses $C084 through $C087 duplicate the functions of the four addresses preceding them, and addresses $C08C through $C08F do also. These eight addresses are explicitly reserved on the Apple IIc and Apple IIc Plus.

$C090–$C0FF

On the Apple II, II Plus, and IIe, each group of 16 addresses of the form $C080 + $n0 is allocated to an interface card (if present) in slot n.

For the Apple IIc family, addresses corresponding to slots 1, 2, 3, 4, and 6 are allocated to a serial interface card, communication interface card, 80-column text card, mouse interface card, and disk controller card, respectively. All other addresses in this range are reserved.

System Monitor

The older models of the Apple II and Apple II Plus included a different version of the System Monitor from the one built into more recent models (and the Apple IIe and IIc). The older version, called the Monitor ROM, had the same standard I/O subroutines as the newer Autostart ROM, but a few of their features were different; for example, there were no arrow keys for vertical cursor motion.

When you start an Apple IIc family member with a DOS or *BASICs* disk and it loads Integer BASIC into the bank-switched area in RAM, it loads the old Monitor along with it. When you type `INT` from Applesoft to activate Integer BASIC, you also activate this copy of the old Monitor, which remains active until you either type `FP` to switch back to Applesoft, which uses the new Monitor in ROM, or activate the 80-column firmware.

I/O in general

Apple IIc–family I/O is different from I/O on the Apple II, II Plus, and IIe in three important respects: the possibility of direct memory access (DMA) transfers, the presence or absence of slots, and the presence or absence of built-in interrupt handling.

DMA transfers

The Apple II, II Plus, and IIe allow DMA transfers, because both the address and the data bus are available at the slots. No true DMA transfer is possible with the Apple IIc family because neither bus is available at any of the back panel connectors.

Slots versus ports

The Apple II and II Plus have eight identical slots; the Apple IIe has seven identical slots plus a 60-pin auxiliary slot for video, add-on memory, and test cards. The Apple IIc family has no slots; instead, each computer has back panel connectors and built-in hardware and firmware that are functional equivalents of slots with cards in them. The back panel connectors are called *ports* for the Apple IIc family.

Interrupts

The original Apple IIc was the first computer in the Apple II series to have built-in interrupt-handling capabilities. The enhanced Apple IIe has very similar interrupt-handling capability included.

The keyboard

Both keyboard layout and character sets vary in the Apple II–series computers. The major keyboard difference in the Apple II series is that the Apple IIe, Apple IIc, and Apple IIc Plus have full ASCII keyboards, while the Apple II and II Plus do not. The extended keyboard version of the Apple IIe includes an 18-key numeric keypad.

Keys, switches, and lights

The Apple II and Apple II Plus computers have identical keyboards with 52 keys (including the Reset switch). The Apple IIe and Apple IIc keyboards have the same 62-key full ASCII keyboard layout, with new and repositioned keys and characters as compared to the Apple II and Apple II Plus. Whereas the Apple II and Apple II Plus have a Repeat key, theApple IIe andApple IIc have an auto-repeat feature built into each character key. The Apple IIc Plus has the same keyboard layout as the Apple Standard Keyboard used by the Apple IIGS and some Macintosh computers; the Apple IIc Plus keyboard does not include a numeric keypad, however.

Some Apple II and Apple II Plus machines have a slide switch inside the case, under the keyboard edge of the cover, for selecting whether or not Reset works without Control. On the Apple IIe and Apple IIc family, there is no choice: Control-Reset works, and Reset alone does not.

The Apple IIc andApple IIe have up arrow and down arrow keys, an Open Apple key (now called *Command*), and a Solid Apple key (now called *Option*); the Apple II and Apple II Plus do not have these four keys. On the extended keyboard Apple IIe and on the Apple IIc Plus, the Solid Apple key is relabeled as the Option key.

The captions on several other keys—Esc, Tab, Control, Shift, Caps Lock, Delete, Return, and Reset—can vary: on the Apple II and Apple II Plus some are abbreviated or missing; on the Apple IIc family all keycaps are lowercase italic; on international models, some captions are replaced by symbols (see Appendix E).

The Apple IIc has two switches that the other models do not have. One switch is for changing between 40-column and 80-column display, the other is for selecting keyboard layout (Sholes versus Dvorak on USA models), or both keyboard layout and character set (on international models). The Apple IIc Plus does not have an 80/40 switch; in its place is the speaker volume control.

The position of the power-on light differs on the Apple II and Apple II Plus, Apple IIe, and Apple IIc family computers. The Apple IIc and Apple IIc Plus have a disk-use light as well.

Character sets

The Apple II and II Plus keyboard character sets are the same. They are described in the *Apple II Reference Manual.*

The Apple IIe and Apple IIc family keyboard character sets are the same: full ASCII. The various keyboard layouts and key assignments for Apple IIc family computers are shown in Chapter 1 and Appendix E of this manual.

To change between the two available keyboard layouts requires modification to the main logic board on the Apple IIe, but only toggling of the keyboard switch on the Apple IIc family.

Apple Computer, Inc., manufactures fully localized models (with regard to power supply and character sets) of both the Apple IIe and the Apple IIc. However, there are minor variations in keyboard layout, even among early and late production models of the same machine. For further details, refer to Appendix E of this manual.

The speaker

The Apple IIc has two speaker features that the three previous models do not have: a two-channel, but monaural, audio output jack for headphones, which disconnects the internal speaker when something is plugged into it, and a volume control. The Apple IIc Plus has a volume control but no audio output jack.

The video display

This section discusses the general differences between Apple IIc family video display capabilities and those of the other computers in the series.

Character sets

The Apple II and II Plus display only uppercase characters, but they display them in three ways: normal, inverse, and flashing. The Apple IIc family and Apple IIe can display uppercase characters in all three ways, and they can display lowercase characters in the normal way. This combination is called the *primary character set*.

The Apple IIc family and Apple IIe have another character set, called the *alternate character set*, that displays a full set of normal and inverse uppercase and lowercase characters, but can't display flashing characters. The primary and alternate character sets are described in Chapter 6. You can switch character sets at any time by means of the AltChar soft switch, also described in Chapter 6.

Flashing display must not be used with the enhanced video firmware active. Use it in 40-column mode with the enhanced video firmware turned off; otherwise, strange displays may result, such as MouseText characters appearing in place of uppercase letters.

To be sure of compatibility with some software, you have to switch the Apple IIc family member's keyboard to uppercase by pressing Caps Lock.

MouseText

The Apple IIc family computers and the enhanced and extended keyboard Apple IIe computers include a set of graphic text characters, called *MouseText characters*, that replace some of the inverse uppercase characters in the alternate character set of the original Apple IIe. The MouseText characters are shown in Figure 6-2, in Chapter 6.

Vertical blanking

A signal called *vertical blanking* indicates when a display device should stop projecting dots until the display mechanism returns from the bottom of the screen to the top to make another pass. During this interval, a program can make changes to display memory pages, and thus provide a smooth, flicker-free transition to a new display.

On the Apple IIe, vertical blanking (VBL) is a signal whose level must be polled. (VBL is not available to software on the Apple II or Apple II Plus.) In the Apple IIc family,there is also a vertical blanking interrupt (VBlInt) that is generated by the IOU on the trailing edge of the active-low VBL signal. Programs intended to run on all Apple II–series computers must take this difference into account.

Display modes

All models have 40-column text mode, Lo-Res graphics mode, Hi-Res graphics mode, and mixed graphics and text modes. The Apple IIe (revision B main logic board) with an Apple IIe Extended 80-Column Text Card, and the Apple IIc family have Double Hi-Res graphics mode also.

Disk I/O

The Apple II, Apple II Plus, and Apple IIe can support up to six disk drives (although four is the recommended maximum) attached to controller cards plugged into slots 6, 5, and 4. The original Apple IIc supports two 5.25-inch disk drives: its built-in drive and one external disk drive. The UniDisk 3.5 Apple IIc supports up to three disk drives: its built-in 5.25-inch disk drive, one external 5.25-inch disk drive, and one external 3.5-inch disk drive. The memory expansion Apple IIc supports up to four disk drives, which can be any combination of its built-in 5.25-inch disk drive, one external 5.25-inch disk drive, and up to three external 3.5-inch disk drives. The Apple IIc Plus supports up to four disk drives, which can be any combination of its built-in 3.5-inch drive, one external Apple 3.5 drive, up to three external UniDisk 3.5 drives, and up to two 5.25-inch disk drives.

Serial I/O

The Apple IIc–family serial ports (ports 1 and 2) are similar to Super Serial Cards installed in slots 1 and 2 of an Apple IIe. The serial port commands are a slightly modified subset of Super Serial Card commands. This subset includes all the commands supported by the earlier Apple Serial Interface Card and Communication Card.

Serial ports versus serial cards

There are several important differences between Apple IIc family serial ports and other Apple II–series computers with serial cards installed in them.

Apple IIc–family serial ports have no switches. Instead, initial values are moved from firmware locations into auxiliary memory when the power is turned on. Changes made to these values in auxiliary memory remain in effect until the power is turned off. Pressing Command-Control-Reset does not change them (because it does not affect auxiliary memory, only main memory).

When the port itself is turned on (with an IN or PR command), the initial values in auxiliary memory are placed in the main memory screen holes assigned to the port. These characteristics can be changed by the port commands. The changed characteristics remain in effect until the port is turned off and then on again (with PR and IN commands).

The command syntax for the Apple IIc–family ports also differs from the syntax for serial cards. A separate command character, Control-A or Control-I, must precede each individual port command, whereas several commands to a serial card can be strung together between the command character and a carriage return character.

The letters used for some of the commands have been changed from those used with the Super Serial Card (such as S instead of B for sending a Break command). To simplify command interpretation, each serial port command letter is unique.

Changing the command character from Control-A to Control-I, or vice versa, makes the Super Serial Card change from communication mode to printer mode and back; this is not the case with Apple *IIc–family serial ports*. With the Apple IIc family, use the system utilities to change modes.

Super Serial Card commands support some functions that Apple IIc–family serial port commands don't support: translating incoming characters, such as changing lowercase to uppercase (for the benefit of the Apple II or II Plus); delaying after sending carriage return, line feed, or form feed, and so on.

Following a Control-I *nnn*N command, the Apple IIc automatically generates a carriage return after *nnn* characters; with the Super Serial Card, you need to turn this feature on with Control-I C.

A complete description of the Apple IIc–family serial port command set is given in Chapter 7.

Serial I/O buffers

The communication port firmware uses auxiliary memory page $08 as an input and output buffer. By doing so, the firmware can keep up with higher baud rates. It can also hide data from the Monitor, Applesoft, and other system software.

Programs written for the Apple IIe or IIc can, of course, store information in auxiliary memory page $08. However, such information is destroyed when the communication port is activated.

Mouse and hand controls

The DB-9 connector on the back panel of the Apple IIc–family computers is used for both the mouse and hand controls. On the Apple IIe, the DB-9 connector supports hand controls only; the mouse must use the connector on the interface card.

Mouse input

The Apple IIc family provides built-in firmware support for a mouse connected to the DB-9 mouse and hand control connector. Apple IIc–family mouse support includes mouse movement and button interrupts (and vertical blanking interrupts for synchronization with the display); Apple IIe mouse support relies on polling VBL instead of vertical blanking interrupts.

As a result of how interrupts are handled on the two machines, the mouse firmware routine calls function somewhat differently for the Apple IIc family and Apple IIe. However, using the calls in the manner described in Chapter 9 and in this section ensures mouse support compatibility between the two machines. The ratio of mouse movement to cursor movement on the Apple IIc family is different from that on the Apple IIe.

The Apple IIe mouse card has a microprocessor on it that regularly polls the mouse to get status and position information. This data is then kept on the card and is available whenever the program requests it through the ReadMouse routine. If the mouse is in transparent mode, your program can read this information at any time. You can use the SerMouse routine to control the conditions upon which the mouse card issues interrupts: any combination of motion in the X direction, motion in the Y direction, and mouse-button activity. When the mouse card issues an interrupt, the CPU passes control to the interrupt handler, which reads the information from where the card processor saved it and puts it in the screen holes.

A program running on an Apple IIe computer with a mouse card is interrupted only under the conditions you have specified, and the data in the screen holes is changed only when the user interrupt handler or your program's polling routine calls ReadMouse. Enabling and inhibiting mouse interrupts does not affect the updating of mouse information by the mouse card's microprocessor.

The Apple IIc mouse does not have a card microprocessor and so the mouse information is collected by the CPU when it receives a mouse interrupt. When the interrupt handler determines that the interrupt was caused by the mouse, the firmware reads the mouse status and updates the screen holes. As for the Apple IIe, whether the interrupt is passed on to your program depends on which interrupts have been enabled using the SetMouse routine. However, because the mouse interrupts the computer's microprocessor, execution of your program is delayed when the mouse is used whether the interrupts are passed on to your program or not. The system interrupt handler not only reads the mouse status from the mouse hardware, but also updates the screen holes with the mouse location, because there is no mouse card on which to temporarily store this data. These delays affect the timing of your program. If your program disables interrupts to preserve program timing, the mouse can never interrupt the processor and so the X and Y values in the screen holes are never updated. In that case, a call to ReadMouse would indicate that there has been no mouse movement.

To minimize the effect on your program of the servicing of mouse interrupts by firmware, the Apple IIc does not generate a mouse interrupt until the mouse has moved twice as far as required by a mouse card in the Apple IIe. To cause mouse behavior on a member of the Apple IIc–family to appear to be the same as that of an Apple IIe mouse, multiply the Apple IIc X and Y values by 2 and set the clamping values to 1/2 those used for the Apple IIe.

With the exception of having to double the amount of mouse movement for Apple IIc–family computers, your assembly-language program can ignore which machine it is running on by obeying the precautions in the following list. If you are using BASIC or Pascal, these conditions are taken care of for you.

- Disable interrupts with an SEI instruction when calling any mouse routine.
- Do not disable interrupts except when absolutely necessary, and be sure to reenable them as soon as possible.
- Do not reenable interrupts after executing the ReadMouse routine until you have read the X and Y mouse motion data from the screen holes..
- Disable interrupts before using the PosMouse or ClampMouse routines to place mouse position information in the screen holes.
- Before executing any mouse routine other than ServeMouse, set the X register to $Cn and set the Y register to $n0, where n is the slot number for the mouse card in the Apple IIe, 4 for the Original and UniDisk 3.5 Apple IIc computers, and 7 for the memory expansion Apple IIc and the Apple IIc Plus
- If your program must disable interrupts for some reason other than to execute a mouse routine, the first call to the ReadMouse routine after you reenable interrupts may give incorrect values. Subsequent calls to ReadMouse will return correct values until the next time you disable and reenable interrupts.

  For example, if your program is reading X values from the mouse while the user is moving it in a direction that would increase values, and interrupts are disabled and reenabled between ReadMouse calls, the X values might change as follows: 6, 7, 8, 9, 8, 9, 10. In practice this slight error in mouse position would probably not be noticed by the user. If you must keep this error from occurring, then do not leave interupts disabled for more then 40 microseconds, or be sure that at least one mouse interrupt has taken place since interrupts were reenabled before calling the ReadMouse routine. Disabling interrupts for mouse calls does not create this problem.

Hand control input and output

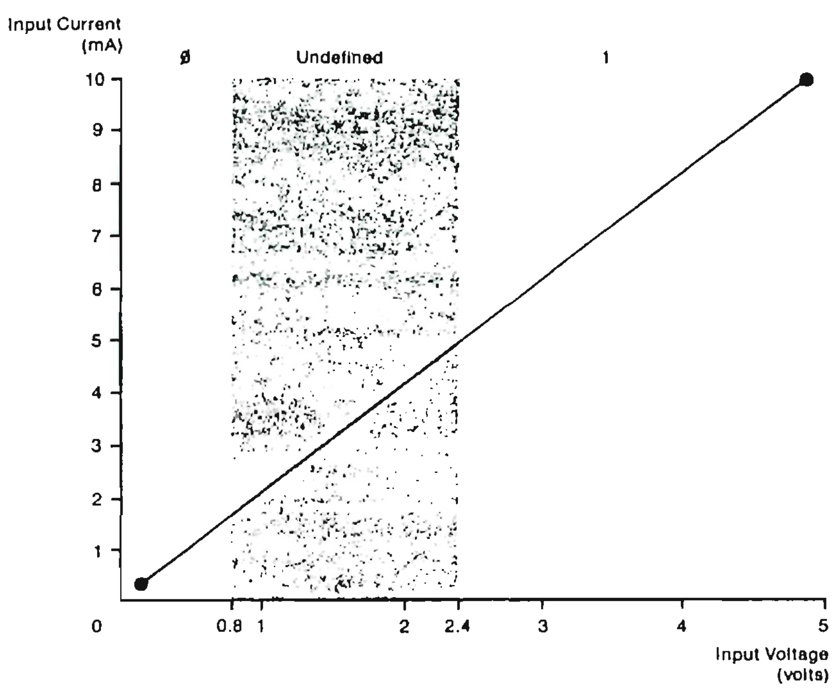
The Apple II, Apple II Plus, and Apple IIe have a 16-pin connector (labeled *GAME I/O*) inside the case that supports three switch inputs, four analog (paddle) inputs, and four annunciator outputs. The Apple IIe and Apple IIc family have a DB-9 back panel connector that supports the three switch inputs and two paddle inputs.

The Apple IIc family does not support the four annunciator outputs.

The characteristic response curve for Apple IIc–family hand controls differs for the from that of the Apple II, II Plus, and IIe. Compare Figure D-1 with Figure 11-45. The response curve is different so the hardware can support identifiable mouse and hand control signals using the same circuits.

The paddle-timing circuit in the Apple II Plus is slightly different from the one in the Apple IIe and Apple IIc family. In the Apple IIe and Apple IIc the 100-ohm fixed resistor is between the NE556 discharge lead and the capacitor; the variable resistor in the paddle is connected directly to the capacitor. In the Apple II Plus, the capacitor is connected directly to the discharge lead, and the fixed resistor is in series with the paddle resistor.

■ **Figure D-1** Apple II, Apple II Plus, and Apple IIe hand control signals



Cassette I/O

The Apple II, II Plus, and IIe all have cassette input and output jacks, memory locations, and Monitor support. The Apple IIc family does not.

Hardware

Besides the different microprocessors used in various models in the Apple II series, there are important differences in power specifications and custom chips.

Power

The power supplies for the Apple II, II Plus, and IIe are essentially the same. Both the external power transformer and internal voltage converter for the Apple IIc have smaller capacity for current and heat dissipation. Therefore, it is important to observe the load limits specified in each of the reference manuals. The Apple IIc Plus computer's power supply is completely internal but otherwise has the same specifications as that of the Apple IIc.

Custom chips

The Apple IIe custom chips (memory management unit and input/output unit) replaced dozens of Apple II Plus chips, and added dozens of new functions. The Apple IIc family also has custom MMU and IOU chips, but they are different ICs from those used in the Apple IIe.

The Apple IIc has four other custom ICs: a general logic unit (GLU), a timing generator (TMG), and a disk controller unit (IWM). The Apple IIc has two hybrid circuits (AUD and VID) for audio and video amplification.

In the Apple IIc Plus, the audio hybrid circuit is replaced by equivalent discrete components on the circuit board. The Apple IIc Plus has two additional custom chips: the multidrive interface glue (MIG) chip, which provides a RAM buffer and hardware interface for 3.5-inch disk drives; and the cache glue gate array (CGGA), which controls a RAM cache and provides a 4 MHz clock for the 65C02.

Appendix E **USA and International Models**

This appendix shows the keyboard layouts and gives the key codes for USA and international models. Some of the keyboard information given in Chapter 5 for the two USA keyboard layouts for the Apple IIc computer is repeated here for easy comparison with the other layouts available.

There are no international versions of the Apple IIc Plus computer. ■

Keyboard layouts and codes

Each of the following subsections has a keyboard illustration and a table of the codes that result from the possible keystrokes. Note, however, that Table E-1 is the basic table of keystrokes and their codes. For simplicity, subsequent tables (up to Table E-7) list only the keystrokes and codes that differ from those in Table E-1.

For example, pressing the A key produces *a* (hexadecimal 61); pressing Shift-A produces uppercase *A* (hexadecimal 41); pressing Control-A or Control-Shift-A produces *SOH* (the ASCII Start Of Header control character, hexadecimal 01). You can tell that this key has the same effect on all keyboards because nothing appears in Tables E-2 through E-7 for that key.

A quick way to find out which characters in the ASCII set change on international keyboards is to check Table E-9. In fact, only a few of them change. The pairing of characters on keys varies more.

◆ *Note:* On all but the French and Italian keyboards, Caps Lock affects only keys that can produce both lowercase letters (with or without an accent) and their uppercase equivalents. With these keys, Caps Lock down is equivalent to holding down Shift, resulting in uppercase instead of lowercase. If a key produces only a lowercase version of an accented letter, then Caps Lock does not affect it.

On the French and Italian keyboards, Caps Lock shifts all the keys. Furthermore, on the French keyboard, when Caps Lock is down the Shift key undoes the shifting.

The shapes and arrangement of keys in Figures E-1 and E-2 follow the ANSI (American National Standards Institute) standard, which is used mainly in North and South America. The shapes and arrangement of keys in Figure E-3 follow the ISO (International Standards Organization) standard used in Europe and elsewhere.

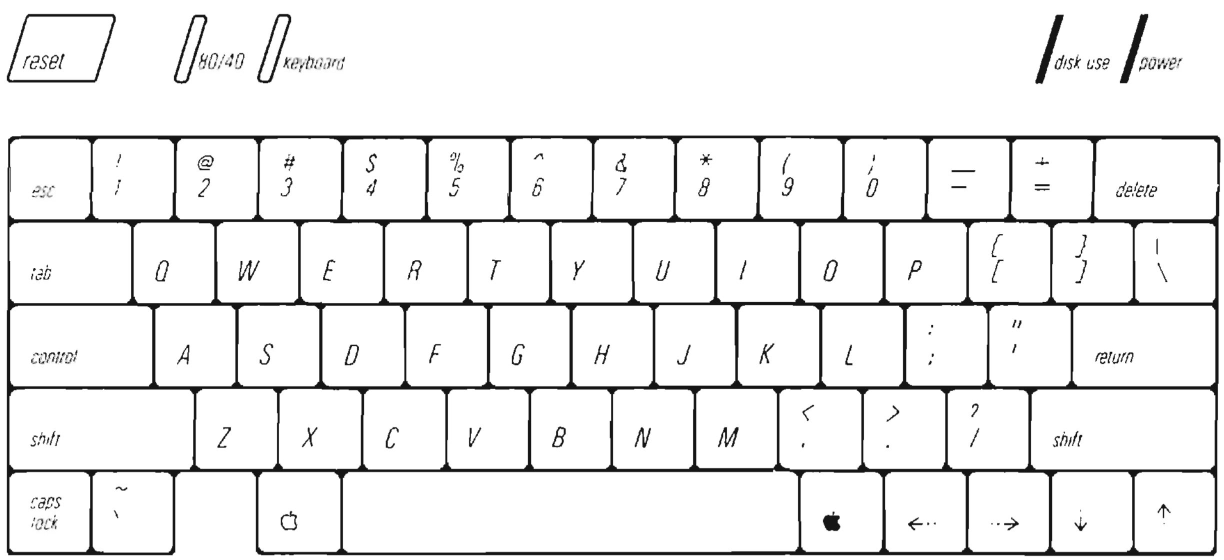
The only differences between the ANSI and ISO versions of the USA keyboard are

- the shapes of three keys: the left Shift key, Caps Lock, and Return
- the position of two keys (| and ~) (compare Figures E-1 and E-3)
- the arrow symbols on Tab, Caps Lock, Return, and the two Shift keys (as shown in Figure E-3)

USA standard (QWERTY) keyboard

Figure E-1 shows the standard (Sholes or QWERTY) keyboard as it is laid out for USA models of the Apple IIc with the keyboard switch up. Table E-1 lists the ASCII codes resulting from all single and combination keystrokes on this keyboard.

■ **Figure E-1** USA standard (QWERTY or Sholes) keyboard, keyboard switch up



■ **Table E-1** Keys and ASCII codes (hexadecimal)

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
Delete	7F	DEL	7F	DEL	7F	DEL	7F	DEL
Left Arrow	08	BS	08	BS	08	BS	08	BS
Tab	09	HT	09	HT	09	HT	09	HT
Down Arrow	0A	LF	0A	LF	0A	LF	0A	LF
Up Arrow	0B	VT	0B	VT	0B	VT	0B	VT
Return	0D	CR	0D	CR	0D	CR	0D	CR
Right Arrow	15	NAK	15	NAK	15	NAK	15	NAK
Esc	1B	ESC	1B	ESC	1B	ESC	1B	ESC
Space	20	SP	20	SP	20	SP	20	SP
' "	27	'	27	'	22	"	22	"

■ **Table E-1** Keys and ASCII codes (hexadecimal) (continued)

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
, <	2C	,	2C	,	3C	<	3C	<
- _	2D	-	1F	US	5F	_	1F	US
. >	2E	.	2E	.	3E	>	3E	>
/ ?	2F	/	2F	/	3F	?	3F	?
0)	30	0	30	0	29	)	29	)
1 !	31	1	31	1	21	!	21	!
2 @	32	2	00	NUL	40	@	00	NUL
3 #	33	3	33	3	23	#	23	#
4 $	34	4	34	4	24	$	24	$
5 %	35	5	35	5	25	%	25	%
6 ^	36	6	1E	RS	5E	^	1E	RS
7 &	37	7	37	7	26	&	26	&
8 •	38	8	38	8	2A	•	2A	•
9 (	39	9	39	9	28	(	28	(
; :	3B	;	3B	;	3A	:	3A	:
= +	3D	=	3D	=	2B	+	2B	+
[{	5B	[	1B	ESC	7B	{	1B	ESC
\ |	5C	\	1C	FS	7C	|	1C	FS
] }	5D	]	1D	GS	7D	}	1D	GS
! ~	60	!	60	!	7E	~	7E	~
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1

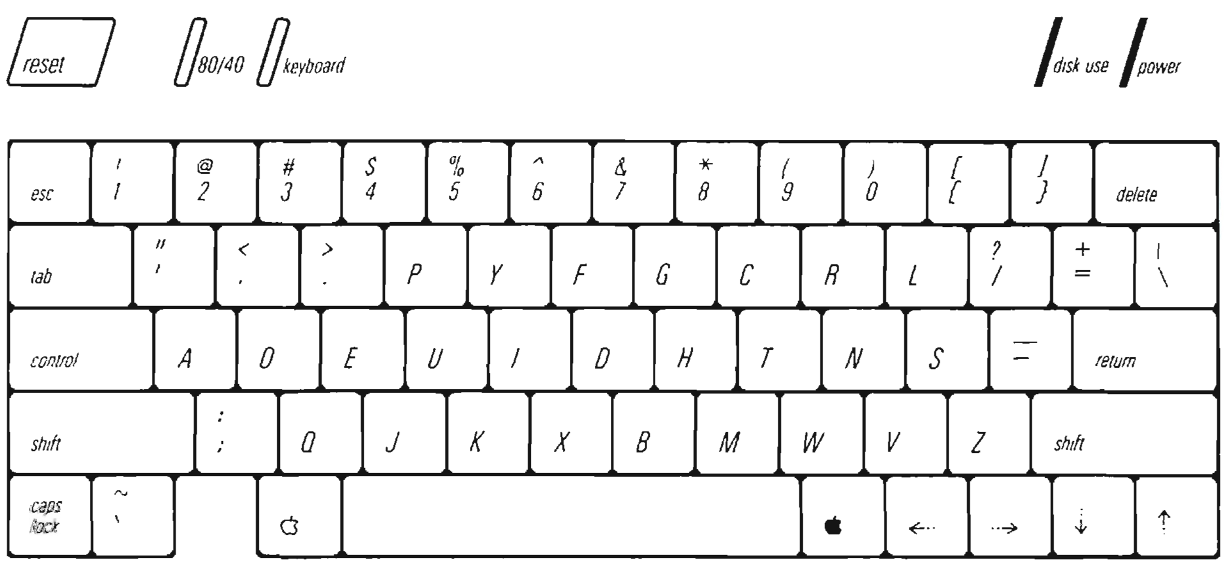
■ **Table E-1** Keys and ASCII codes (hexadecimal) (continued)

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB
X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

USA simplified (Dvorak) keyboard

Figure E-2 shows the Dvorak layout of the USA keyboard. This is the layout of keys that is in effect when the keyboard switch is in its down position. All keystrokes produce the same ASCII codes as those shown in Table E-1.

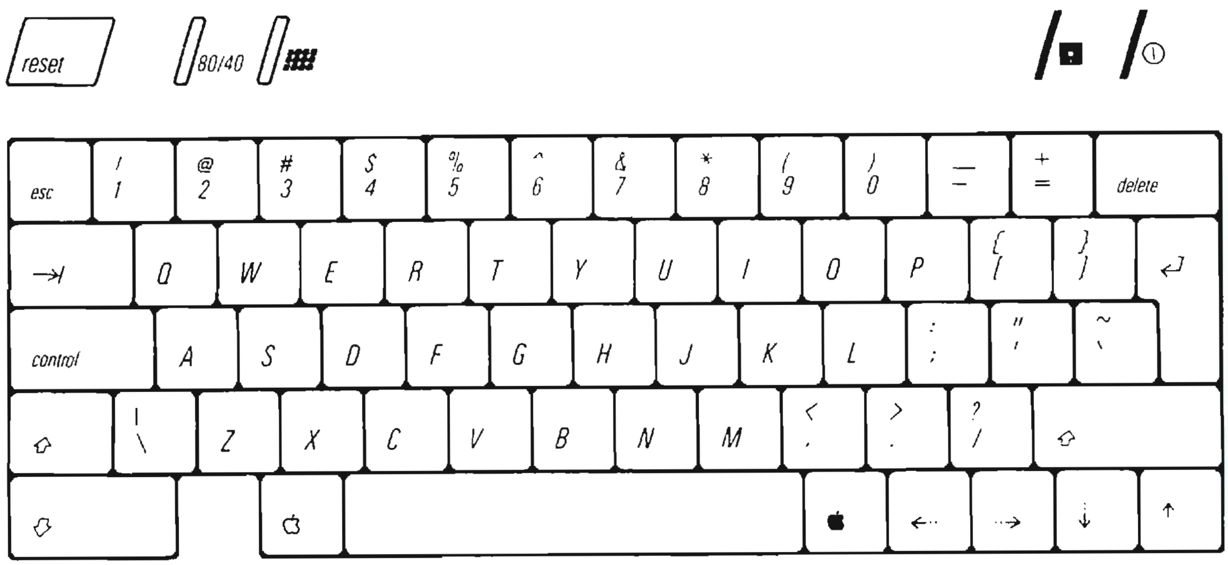
■ **Figure E-2** USA simplified (or Dvorak) keyboard, keyboard switch down; shaded characters may be in different positions on some models.



ISO layout of USA keyboard

Figure E-3 shows the layout of all ISO European keyboards (except the Italian keyboard) when the keyboard switch is up. All keystrokes produce the same ASCII codes as those shown in Table E-1.

■ **Figure E-3** ISO version of USA standard keyboard, keyboard switch up



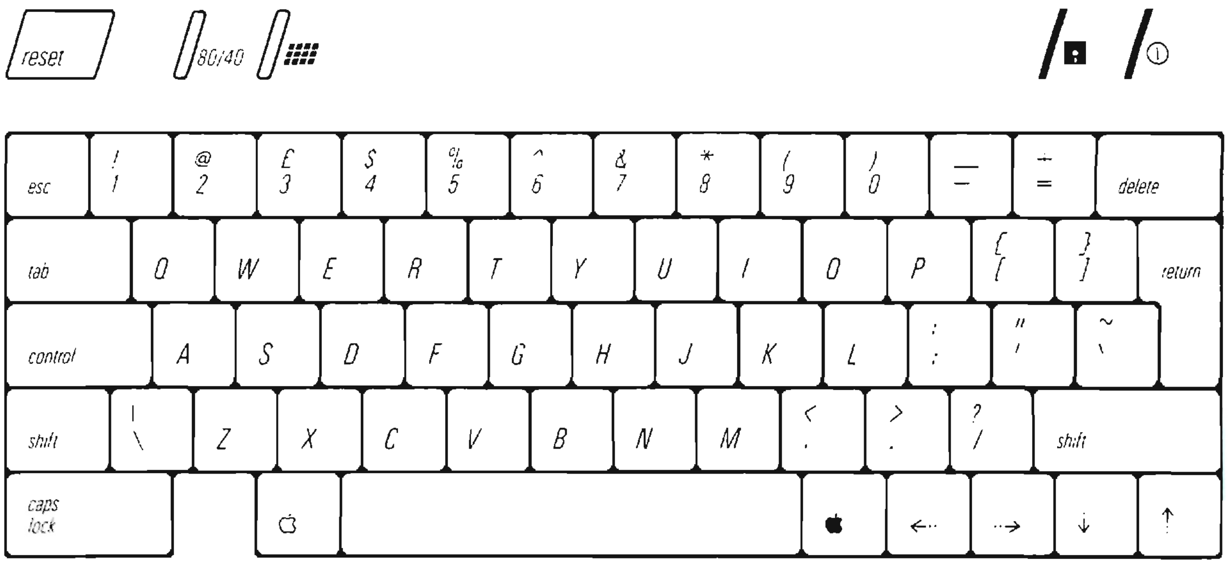
British keyboard

With the keyboard switch up, the British model of the Apple IIc keyboard layout is as shown in Figure E-3, and keystrokes produce the ASCII codes shown in Table E-1.

With the keyboard switch down, the British model keyboard layout is as shown in Figure E-4. The change in ASCII code production (from that in Table E-1) is shown in Table E-2.

The only changed character is the substitution of the British pound-sterling symbol (£) for the number symbol (#) on the shifted 3-key.

■ **Figure E-4** British keyboard, keyboard switch down



■ **Table E-2** British keyboard code differences from Table E-1

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
3£	33	3	33	3	23	£	23	£

Note: Codes are in hexadecimal.

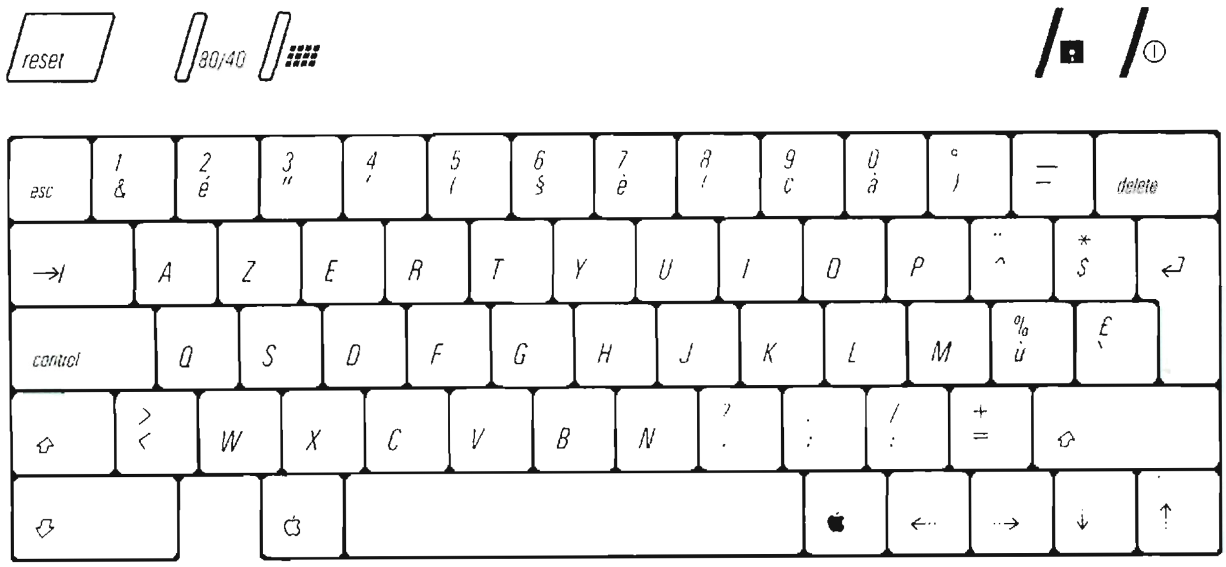
French keyboard

With the keyboard switch up, the French model of the Apple IIc keyboard layout is as shown in Figure E-3, and keystrokes produce the ASCII codes shown in Table E-1.

With the keyboard switch down, the French model keyboard layout is as shown in Figure E-5. The changes in ASCII code production (from that in Table E-1) are shown in Table E-3.

Note that on the French keyboard, Caps Lock shifts to the upper characters on all keys. With Caps Lock on, Shift "unshifts" to the lower character on any key pressed with it.

■ **Figure E-5** French keyboard, keyboard switch down



■ **Table E-3** French keyboard code differences from Table E-1

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
& 1	26	&	26	&	31	1	31	1
é 2	7B	é	7B	é	32	2	32	2
" 3	22	"	22	"	33	3	33	3
' 4	27	'	27	'	34	4	34	4
(5	28	(	28	(	35	5	35	5
§ 6	5D	§	1D	GS	36	6	1D	GS
è 7	7D	è	7D	è	37	7	37	7
! 8	21	!	21	!	38	8	38	8
ç 9	5C	ç	1C	FS	39	9	1C	FS
à 0	40	à	00	NUL	30	0	00	NUL
) °	29	)	1B	ESC	5B	°	1B	ESC
^ ¨	5E	^	1E	RS	7E	¨	1E	RS

■ **Table E-3** French keyboard code differences from Table E-1 (continued)

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
$ *	24	$	24	$	2A	*	2A	*
ù %	7C	ù	7C	ù	25	%	25	%
` £	60	`	60	`	23	£	23	£
< >	3C	<	3C	<	3E	>	3E	>
, ?	2C	,	2C	,	3F	?	3F	?
; .	3B	;	3B	;	2E	.	2E	.
: /	3A	:	3A	:	2F	/	2F	/

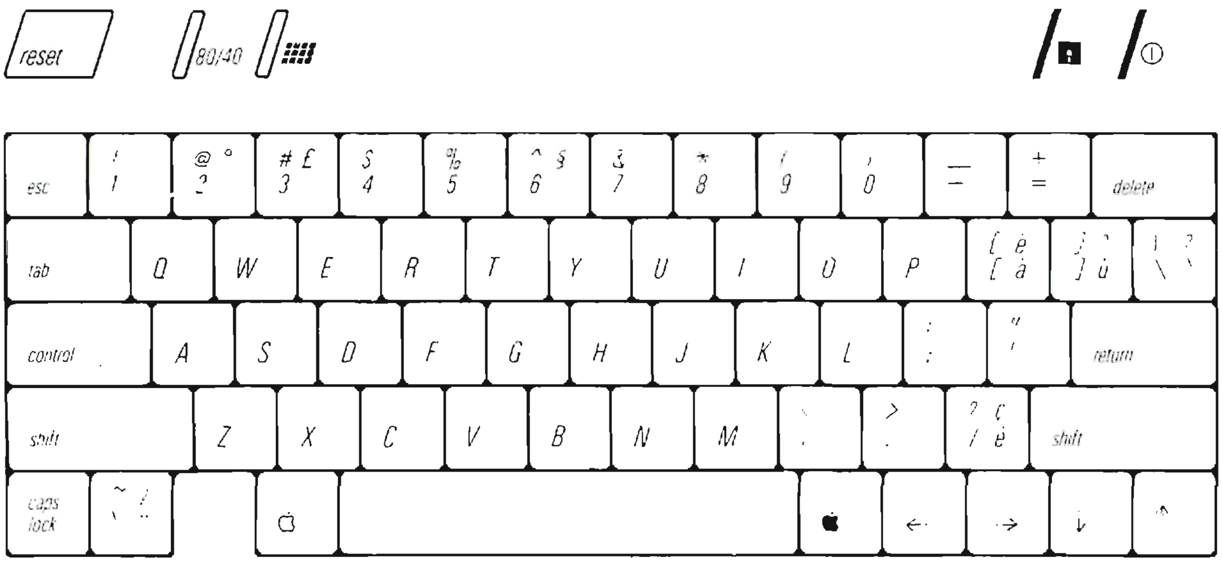
Note: Codes are in hexadecimal.

Canadian keyboard

With the keyboard switch up, the Canadian model of the Apple IIc keyboard layout is as shown in Figure E-1, and keystrokes produce the ASCII codes shown in Table E-1.

With the keyboard switch down, the Canadian model keyboard layout is as shown in Figure E-6. The changes in ASCII code production (from that in Table E-1) are shown in Table E-4.

■ **Figure E-6** Canadian keyboard, keyboard switch down



■ **Table E-4** Canadian keyboard code differences from Table E-1

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
2°	32	2	00	NUL	5B	°	00	NUL
3£	33	3	33	3	23	£	23	£
6§	36	6	RS	1E	5D	§	RS	1E
àè	40	à	7F	DEL	7D	è	7F	DEL
Ù ^	7C	Ù	7C	Ù	5E	^	5E	^
\`?	60	\`	ESC	1B	3F	?	1D	GS
éç	7B	é	1C	FS	5C	ç	1C	FS
¨/	7E	¨	7E	¨	2F	/	2F	/

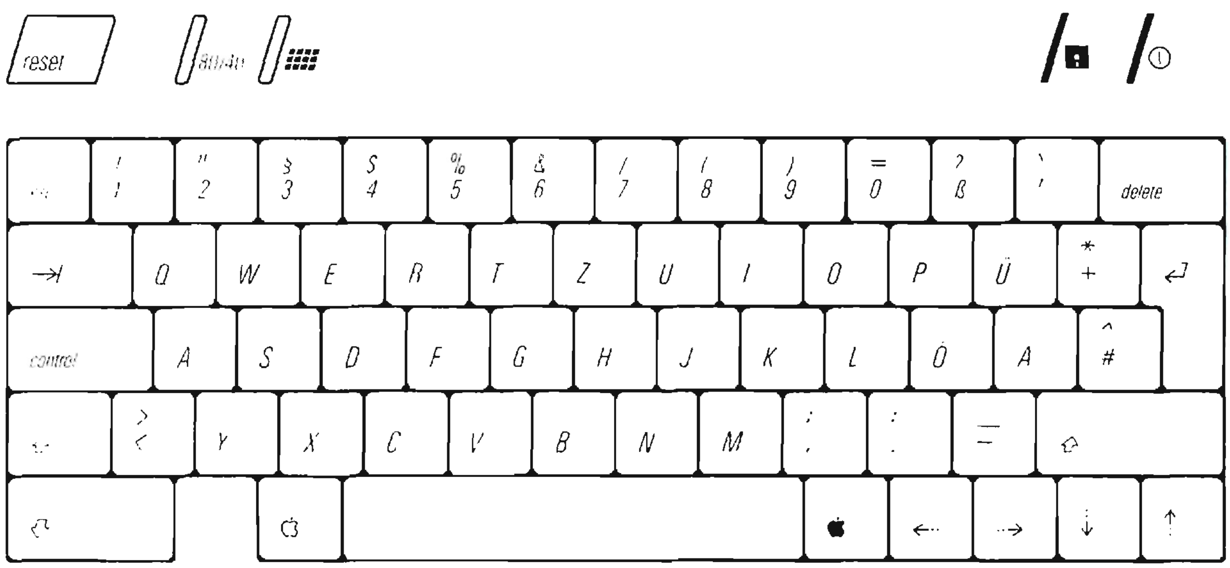
Note: Codes are in hexadecimal.

German keyboard

With the keyboard switch up, the German model of the Apple IIc keyboard layout is as shown in Figure E-3, and keystrokes produce the ASCII codes shown in Table E-1.

With the keyboard switch down, the German model keyboard layout is as shown in Figure E-7. The change in ASCII code production (from that in Table E-1) is shown in Table E-5.

■ **Figure E-7** German keyboard, keyboard switch down



■ **Table E-5** German keyboard code differences from Table E-1

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
2 "	32	2	32	2	22	"	22	"
3 §	33	3	00	NUL	40	§	00	NUL
6 &	36	6	36	6	26	&	26	&
7 /	37	7	37	7	2F	/	2F	/

■ **Table E-5** German keyboard code differences from Table E-1 (continued)

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
8(	38	8	38	8	28	(	28	(
9)	39	9	39	9	29	)	29	)
0 =	30	0	30	0	3D	=	3D	=
ß ?	7E	ß	7E	ß	3F	?	3F	?
Ü	7D	ü	1D	GS	5D	Ü	1D	GS
+ *	2B	+	2B	+	2A	*	2A	*
Ö	7C	ö	1C	FS	5C	Ö	1C	FS
Ä	7B	ä	1B	ESC	5B	Ä	1B	ESC
# ^	23	#	1E	RS	5E	^	1E	RS
< >	3C	<	3C	<	3E	>	3E	>
, ;	2C	,	2C	,	3B	;	3B	;
. :	2E	.	2E	.	3A	:	3A	:

Note: Codes are in hexadecimal.

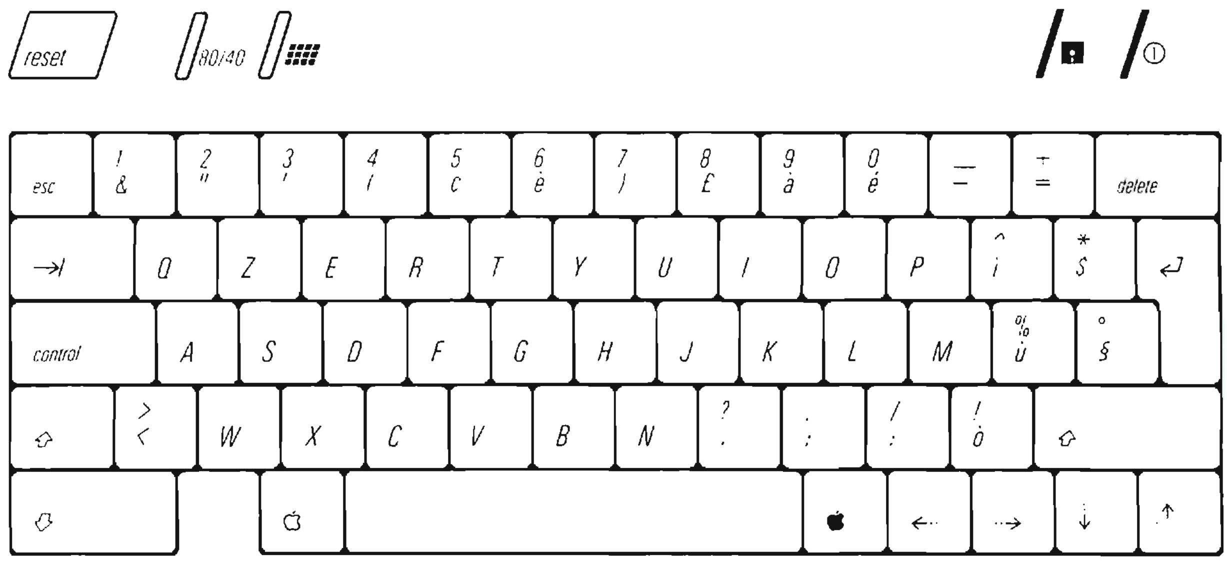
Italian keyboard

With the keyboard switch up, the Italian model of the Apple IIc keyboard layout is as shown in Figure E-3, and keystrokes produce the ASCII codes shown in Table E-1.

With the keyboard switch down, the Italian model keyboard layout is as shown in Figure E-8. The change in ASCII code production (from that in Table E-1) is shown in Table E-6.

With the keyboard switch up, the Italian model keyboard produces exactly the same ASCII codes for each key, but what is displayed differs for the ten characters indicated with the circled numbers 0, 2–5, and 7–11 in Table E-8.

■ **Figure E-8** Italian keyboard, keyboard switch down



■ **Table E-6** Italian keyboard code differences from Table E-1

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
& 1	26	&	26	&	31	1	31	1
" 2	22	"	22	"	32	2	32	2
' 3	27	'	27	'	33	3	33	3
(4	28	(	28	(	34	4	34	4
ç 5	5C	ç	1C	FS	35	5	1C	FS
è 6	7D	è	7D	è	36	6	36	6
) 7	29	)	29	)	37	7	37	7
£ 8	23	£	23	£	38	8	38	8
à 9	7B	à	7B	à	39	9	39	9
é 0	5D	é	1D	GS	30	0	1D	GS
ì ^	7E	ì	1E	RS	5E	^	1E	RS
$ *	24	$	24	$	2A	*	2A	*
ù %	60	ù	60	ù	25	%	25	%
§ °	40	§	00	NUL	5B	°	1B	ESC
< >	3C	<	3C	<	3E	>	3E	>
, ?	2C	,	2C	,	3F	?	3F	?
; .	3B	;	3B	;	2E	.	2E	.
: /	3A	:	3A	:	2F	/	2F	/
ò !	7C	ò	7C	ò	21	!	21	!

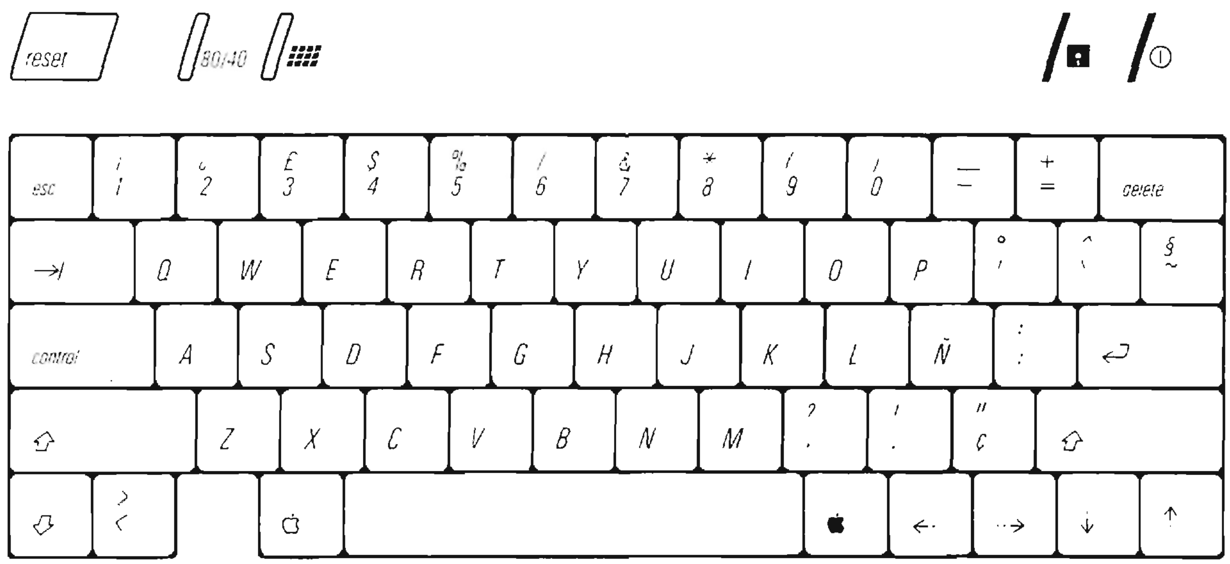
Note: Codes are in hexadecimal.

Western Spanish keyboard

With the keyboard switch up, the Western (that is, American) Spanish model of the Apple IIc keyboard layout is as shown in Figure E-1, and keystrokes produce the ASCII codes shown in Table E-1.

With the keyboard switch down, the Western Spanish model keyboard layout is as shown in Figure E-9. The change in ASCII code production (from that in Table E-1) is shown in Table E-7.

■ **Figure E-9** Western Spanish keyboard, keyboard switch down



■ **Table E-7** Western Spanish keyboard code differences from Table E-1

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
1 ¡	31	1	31	1	5B	¡	5B	¡
2 ¿	32	2	32	2	5D	¿	5D	¿
3 £	33	3	33	3	23	£	23	£
6 /	36	6	36	6	2F	/	2F	/
´ °	27	´	27	´	7B	°	7B	°
` ^	60	`	00	NUL	5E	^	00	NUL
~ §	7E	~	7F	DEL	40	§	7F	DEL
Ñ	7C	ñ	1C	FS	5C	Ñ	1C	FS

■ **Table E-7** Western Spanish keyboard code differences from Table E-1 (continued)

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
,?	2C	,	2C	,	3F	?	3F	?
.!	2E	.	2E	.	21	!	21	!
ç"	7D	ç	1D	GS	22	"	1D	GS
< >	3C	<	1E	RS	3E	>	1E	RS

Note: Codes are in hexadecimal.

ASCII character sets

Table E-8 lists the ASCII (American National Standard Code for Information Interchange) codes that the Apple IIc family uses for the English (USA) character set. Characters that are different in other character sets are marked with an asterisk; the alternate characters are shown in Table E-9.

■ **Table E-8** ASCII codes for the English (USA) character set

ASCII char	Dec code	Hex code	ASCII char	Dec code	Hex code	ASCII char	Dec code	Hex code	ASCII char	Dec code	Hex code
NUL	00	00	SP	32	20	@*	64	40	`*	96	60
SOH	01	01	!	33	21	A	65	41	a	97	61
STX	02	02	"	34	22	B	66	42	b	98	62
ETX	03	03	#*	35	23	C	67	43	c	99	63
EOT	04	04	$	36	24	D	68	44	d	100	64
ENQ	05	05	%	37	25	E	69	45	e	101	65
ACK	06	06	&	38	26	F	70	46	f	102	66
BEL	07	07	'	39	27	G	71	47	g	103	67
BS	08	08	(	40	28	H	72	48	h	104	68
HT	09	09	)	41	29	I	73	49	i	105	69
LF	10	0A	*	42	2A	J	74	4A	j	106	6A
VT	11	0B	+	43	2B	K	75	4B	k	107	6B
FF	12	0C	,	44	2C	L	76	4C	l	108	6C
CR	13	0D	-	45	2D	M	77	4D	m	109	6D
SO	14	0E	.	46	2E	N	78	4E	n	110	6E
SI	15	0F	/	47	2F	O	79	4F	o	111	6F
DLE	16	10	0	48	30	P	80	50	p	112	70
DC1	17	11	1	49	31	Q	81	51	q	113	71
DC2	18	12	2	50	32	R	82	52	r	114	72
DC3	19	13	3	51	33	S	83	53	s	115	73
DC4	20	14	4	52	34	T	84	54	t	116	74
NAK	21	15	5	53	35	U	85	55	u	117	75
SYN	22	16	6	54	36	V	86	56	v	118	76
ETB	23	17	7	55	37	W	87	57	w	119	77
CAN	24	18	8	56	38	X	88	58	x	120	78
EM	25	19	9	57	39	Y	89	59	y	121	79
SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A
ESC	27	1B	;	59	3B	[*	91	5B	{*	123	7B
FS	28	1C	<	60	3C	*	92	5C	|*	124	7C
GS	29	1D	=	61	3D	]*	93	5D	}*	125	7D
RS	30	1E	>	62	3E	^	94	5E	~*	126	7E
US	31	1F	?	63	3F	_	95	5F	DEL	127	7F

* The characters marked with an asterisk are not the same in all character sets. The differences are shown in Table E-9.

■ **Table E-9** Differences from English (USA) character set

	Hexadecimal ASCII Codes									
	23	40	5B	5C	5D	60	7B	7C	7D	7E
English (USA)	#	@	[	\	|	`	{	|	}	~
English (UK)	£	@	[	\	]	`	{	|	}	~
German	#	§	Ä	Ö	Ü	`	ä	ö	ü	ß
French	£	à	°	ç	§	`	é	Ù	è	¨
Italian	£	§	°	ç	é	Ù	à	ò	è	ì
Spanish	£	§	¡	Ñ	¿	`	°	ñ	ç	~

Power supply specifications

The basic specifications of the power supply furnished with the Apple IIc for use in Europe and other countries having 50 Hz alternating current are shown in Table E-10.

■ **Table E-10** 50 Hz power supply specifications

Line voltage	199 to 255 VAC, 50 Hz
Maximum input power consumption	25 W
Supply voltage	+15 VDC (nominal)
Supply current	1.2 A (nominal)

Appendix F **Firmware Entry Points**

This appendix lists and describes the firmware entry points available for your use in the Apple IIc family of computers. Only the entry points described in this appendix are supported by Apple Computer, Inc.; if you use any other firmware routines or entry points for firmware routines, your program will almost certainly fail the next time the firmware is revised.

The firmware routines in this appendix are presented in sequence of their entry points. For your convenience, Table F-1 lists the routines alphabetically by name and briefly summarizes the function of each.

Firmware routines for enhanced video firmware I/O are described in the section "I/O Firmware Support for Video Output" in Chapter 6. ■

■ **Table F-1** Summary of firmware entry points

Routine	Entry point	Function
Advance	$FBF4	Advance the cursor
AppleII	$FB60	Clear text screen and display name of machine
BasCalc	$FBC1	Calculate base address for a line on the text screen
Bell	$FF3A	Generate a tone for 0.1 second preceded by a 0.01 second delay
Bell1	$FBDD	Send a bell character to standard output
Bell1.2	$FBE2	Generate a tone for 0.1 second with no delay
Bell2	$FBE4	Generate a tone for a specified amount of time
Break	$FA4C	6502 break handler
BS	$FC10	Backspace the cursor
ClrEOL	$FC9C	Clear line of text from cursor to end of line
ClrEOLZ	$FC9E	Clear line of text from column in Y register to end of line
ClrEOP	$FC42	Clear text window from cursor to bottom of window
ClrScr	$F832	Clear the Lo-Res graphics display to black
ClrTop	$F836	Clear the top 40 lines of the Lo-Res graphics display
COut	$FDED	Call the standard output routine
COut1	$FDF0	Send text to the screen; default routine used for standard output
COutZ	$FDF6	Alternate entry point to COut1
CR	$FC62	Move cursor to beginning of next line
CROut	$FD8E	Send a carriage return to standard output
CROut1	$FD8B	Clear to end of line and send carriage return to standard output
Dig	$FF8A	Convert an ASCII number into a hexadecimal digit and store in A2L/A2H
FD10	$FD10	Jump to standard input
GBasCalc	$F847	Calculate the base address for a row on the Lo-Res graphics screen
GetLn	$FD6A	Display standard prompt and read a line of text
GetLn0	$FD6C	Display prompt in A register and read a line of text
GetLn1	$FD6F	Read a line of text
GetLnZ	$FD67	Send CR to standard output, display standard prompt, and read a line of text
GetNum	$FFA7	Scan input buffer; decode ASCII numbers and store them in A2L/A2H
Go	$FEB6	Set registers to stored values and jump to address in A1L/A1H
HeadR	$FCC9	Obsolete entry point
HLine	$F819	Draw a horizontal line of blocks on the Lo-Res graphics screen
Home	$FC58	Move cursor to upper left corner and clear to bottom of text window

■ **Table F-1** Summary of firmware entry points (continued)

Routine	Entry Point	Function
IDRoutine	$FE1F	Apple IIGS ID routine
Init	$FB2F	Initialize the text screen
InPort	$FE8B	Set input hooks to point to a specified port
InsDs1.2	$F88C	Load A register with an opcode and call InsDs2
InsDs2	$F88E	Determine the length of a 6502 instruction
InstDsp	$F8D0	Disassemble and display an instruction
IORTS	$FF58	Known RTS instruction
KbdWait	$FB88	Wait for a keypress
KeyIn	$FD1B	Read the keyboard; default routine used for standard input
KeyIn0	$FD18	Jump to standard input
LF	$FC66	Move cursor down one line
List	$FE5E	Disassemble and print 20 instructions
Mon	$FF65	Sound speaker and enter the Monitor
Monz	$FF69	Enter the Monitor; standard Monitor entry point
Monz4	$FF70	Enter the Monitor after your program has read a line of text
Move	$FE2C	Copy the contents of memory from one range of locations to another
NewBrk	$FA47	Alternate entry point to BREAK
NxtA1	$FCBA	Compare A1L/A1H with A2L/A2H and increment A1L/A1H
NxtA4	$FCB4	Increment A4L/A4H
NxtChr	$FFAD	Test character in the input buffer and send to Dig if it is a number
NxtCol	$F85F	Increment the value of COLOR by 3
OldBrk	$FA59	Break routine called by BREAK
OldIRQ	$FA40	Jump to interrupt-handling routine
OldRst	$FF59	Initialize input, output, and text screen, and enter Monitor
OutPort	$FE95	Set output hooks to point to a specified port
PCAdj	$F953	Load the Monitor's PC into the A and Y registers, and increment the value
Plot	$F800	Plot a block on the Lo-Res graphics screen at the location you specify
Plot1	$F80E	Plot a block on the Lo-Res graphics screen on the current line
PrA1	$FD92	Send a CR, contents of A1L/A1H, and a hyphen to standard output
PrBl2	$F94A	Print the number of blank spaces specified by the X register
PrBlnk	$F948	Print three blank spaces
PrByte	$FDDA	Print the contents of the A register

■ **Table F-1** Summary of firmware entry points (continued)

Routine	Entry Point	Function
PRead	$FB1E	Read the dial on a hand control
PRead4	$FB21	Alternate entry point of PRead
PrErr	$FF2D	Print "ERR" and send a bell character to standard output
PrHex	$FDE3	Print the lower nibble of the A register
PrntAX	$F941	Print the contents of the A and X registers
PrntX	$F944	Print the contents of the X register
PrntYX	$F940	Print the contents of the Y and X registers
PwrUp	$FAA6	Cold-start initialization routine
RdChar	$FD35	Set Escape mode to active and jump to RdKey
RdKey	$FD0C	Place character in A register and jump to standard input
Read	$FEFD	Obsolete entry point
RegDsp	$FAD7	Display Monitor register contents stored in memory
Reset	$FA62	Reset handler
Restore	$FF3F	Set registers to stored values
Save	$FF4A	Store in memory the contents of the registers
Scrn	$F871	Return the color value of a block on the Lo-Res graphics display
Scroll	$FC70	Scroll characters in text window up one line
SetCol	$F864	Set the color used for plotting in the Lo-Res graphics mode
SetGr	$FB40	Set display to mixed graphics mode
SetInv	$FE80	Set inverse flag to display inverse characters
SetKbd	$FE89	Set input hooks to point to KeyIn
SetNorm	$FE84	Set inverse flag to display normal characters
SetPwrC	$FB6F	Set Validity-Check byte
SetTxt	$FB39	Initialize text screen; don't force text Page 1
SetVid	$FE93	Set output hooks to point to COut1
SetWnd	$FB4B	Set the size of the text window
StorAdv	$FBF0	Place a printable character on the text screen
TabV	$FB5B	Vertical tab to line specified in A register; update CV
Up	$FC1A	Move the cursor up one line

■ **Table F-1** Summary of firmware entry points (continued)

Routine	Entry Point	Function
Verify	$FE36	Compares the contents of two ranges of memory
Version	$FBB3	Identification byte
VidOut	$FBFD	Send a character to StorAdv; handle CR, LF, BS, and BEL characters
VidWait	$FB78	Pause output when Control-S is pressed
VLine	$F828	Draw a vertical line of blocks on the Lo-Res graphics screen
VTab	$FC22	Vertical tab to line specified by CV
VTabZ	$FC24	Vertical tab to line specified in A register; don't update CV
Wait	$FCA8	Delay loop
Write	$FECD	Obsolete entry point
ZIDByte	$FBC0	Identification byte
ZIDByte2	$FBBF	Identification byte
ZMode	$FFC7	Store $00 in Monitor's Mode byte

$F800 Plot

Description

This routine puts a single block of the color value set by SetCol on the Lo-Res graphics display screen at the location you specify.

Input

Registers
A = block's vertical position ($00–$2F)
X = any value
Y = block's horizontal position ($00–$27)

Memory
COLOR (address $30) = color value set by SetCol

Output

Registers

A = undefined
X = unchanged
Y = unchanged
P = undefined

See also

SetCol
"Lo-Res Graphics" in Chapter 6
"Display Page Maps" in Chapter 6

$F80E Plot1

Description

This routine puts a single block of the color value set by SetColon the Lo-Res graphics display screen at the horizontal position you specify on the current row. The current row is determined by the values stored in locations GBASL/GBASH.

Use Plot if you want to input both the vertical and horizontal positions. Use GBasCalc to set GBASL and GBASH for a particular vertical position.

Input

Registers
A = any value
X = any value
Y = block's horizontal position ($00–$27)

Memory
COLOR (address $30) = color value set by SetCol

Output

Registers
A = undefined
X = unchanged
Y = unchanged
P = undefined

See also

GBasCalc
Plot
SetCol
"Lo-Res Graphics" in Chapter 6
"Display Page Maps" in Chapter 6

$F819 HLine

Description

This routine draws a horizontal line of blocks on the Lo-Res graphics display screen. The color of the blocks is determined by COLOR, which you can set with the SetCol routine.

Input

Registers
A = block's vertical position ($00–$2F)
X = any value
Y = horizontal position of leftmost block ($00–$27)

Memory
H2 (address $2C) = horizontal position of rightmost block ($00–$27)
COLOR (address $30) = color value set by SetCol

Output

Registers
A = undefined
X = unchanged
Y = undefined
P = undefined

See also

SetCol
"Lo-Res Graphics" in Chapter 6
"Display Page Maps" in Chapter 6

$F828 VLine

Description

This routine draws a vertical line of blocks on the Lo-Res graphics display screen. The color of the blocks is determined by COLOR, which you can set with the SetCol routine.

Input

Registers
A = vertical position of top block ($00–$2F)
X = any value
Y = block's horizontal position ($00–$27)

Memory
V2 (address $2D) = vertical position of bottom block ($00–$2F)
COLOR (address $30) = color value set by SetCol

Output

Registers
A = undefined
X = unchanged
Y = undefined
P = undefined

See also

SetCol
"Lo-Res Graphics" in Chapter 6
"Display Page Maps" in Chapter 6

$F832 ClrScr

Description

This routine clears the Lo-Res graphics display to black.

◆ *Note:* If you call this routine while the video display is in text mode, the routine fills the screen with inverse at-sign (@) characters.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = undefined
X = unchanged
Y = undefined
P = changed

See also

"Lo-Res Graphics" in Chapter 6

$F836 ClrTop

Description This routine clears the top 40 lines of the Lo-Res graphics display to black. You can use this routine to clear the graphics portion of the mixed-mode screen.

Input Registers A = any value
X = any value
Y = any value

Output Registers A = undefined
X = unchanged
Y = undefined
P = changed

See also "Mixed-Mode Displays" in Chapter 6.

$F847 GBasCalc

Description This routine calculates the starting address in memory (the *base address*) of the data for a specific row on the Lo-Res graphics screen. This address is stored at GBASL/GBASH.

Input Registers
A = vertical position of the row ($00–$2F)
X = any value
Y = any value

Output Registers
A = the value stored in GBASL
X = unchanged
Y = unchanged
P = undefined

See also Plot1
"Graphics Modes" in Chapter 6
"Display Page Maps" in Chapter 6

$F85F NxtCol

Description

This routine adds 3 to the current color used for Lo-Res graphics. Use SetCol to set a specific color for Lo-Res graphics.

Input

Registers	A = any value X = any value Y = any value	
Memory	COLOR (address $30) = current color value	

Output

Registers	A = new value for color, same number in both nibbles X = unchanged Y = unchanged P = undefined	
Memory	COLOR (address $30) = new color value	

See also

SetCol
"Lo-Res Graphics" in Chapter 6

$F864 SetCol

Description This routine sets the color used for plotting in the Lo-Res graphics mode. The colors are shown in Table F-2.

■ **Table F-2** Lo-Res graphics colors

Nibble value		Color
Dec	**Hex**	
0	$00	Black
1	$01	Magenta
2	$02	Dark blue
3	$03	Purple
4	$04	Dark green
5	$05	Gray 1
6	$06	Medium blue
7	$07	Light blue
8	$08	Brown
9	$09	Orange
10	$0A	Gray 2
11	$0B	Pink
12	$0C	Light green
13	$0D	Yellow
14	$0E	Aquamarine
15	$0F	White

Note: Colors may vary, depending on adjustment of monitor or television set.

Input Registers A = new color in low nibble; high nibble can be any value
X = any value
Y = any value

Output	Registers	A = new color; same number in both nibbles X = unchanged Y = unchanged P = undefined
	Memory	COLOR (address $30) = new value for color
See also	"Lo-Res Graphics" in Chapter 6	

$F871 Scrn

Description This routine returns the color value of a single block on the Lo-Res graphics display.

Input Registers A = block's vertical position ($00–$2F)
X = any value
Y = block's horizontal position ($00–$27)

Output Registers A = color of specified block in low nibble; high nibble = 0
X = unchanged
Y = unchanged
P = undefined

See also "Lo-Res Graphics" in Chapter 6

$F88C InsDs1.2

Description

The instruction at this address loads the A register with an opcode, the length of which is calculated by the InsDs2 routine. The opcode is loaded by an `LDA (PCL,X)` instruction. Immediately after executing this instruction, the CPU executes the InsDs2 routine.

Input

Registers	A = any value X = offset into buffer that contains pointers to instructions Y = any value	
Memory	PCL/PCH (addresses \$3A–\$3B) = address of start of buffer	

Output

Registers	A = opcode for evaluation by InsDs2 X = undefined Y = \$00 P = undefined	
Memory	LENGTH (address \$2F) = length – 1 of 6502 instruction, or = \$00 if the A register does not contain a 6502 opcode	

See also

InsDs2

$F88E InsDs2

Description

This routine determines the length (minus 1) of the 6502 instruction specified by the opcode stored in the A register (accumulator). You can use the InsDs1.2 entry point to load an opcode into the A register.

△ **Important** The InsDs2 routine returns the correct instruction length for 6502 opcodes only. All other opcodes return a length of $00.

The BRK opcode returns a length of $00. △

Input

Registers
A = opcode to be evaluated
X = any value
Y = any value

Output

Registers
A = undefined
X = undefined
Y = $00
P = undefined

Memory
LENGTH (address $2F) = length – 1 of 6502 instruction, or = $00 if the A register does not contain a 6502 opcode

See also

InsDs1.2

$F8D0 InstDsp

Description

This routine disassembles and prints to standard output the instruction pointed to by the address in PCL/PCH.

Input

Registers
A = any value
X = any value
Y = any value

Memory
PCL/PCH (addresses $3A–$3B) = pointer to opcode

Output

Registers
A = undefined
X = undefined
Y = undefined
P = changed

See also

"Standard Output Routines" in Chapter 4
"Disassembled Programs" in Chapter 10

$F940 PrntYX

Description

This routine calls the PrByte routine to print to standard output the contents of the Y and X registers as a four-digit hexadecimal value. The X register is printed as the low byte and the Y register is the high byte.

Input

Registers
A = any value
X = low byte of the value to be printed
Y = high byte of the value to be printed

Output

Registers
A = undefined
X = unchanged
Y = unchanged
P = changed

See also

PrByte
"Standard Output Routines" in Chapter 4

$F941 PrntAX

Description

This routine prints to standard output the contents of the A and X registers as a four-digit hexadecimal value. The X register is printed as the low byte and the A register is the high byte.

Input

Registers
A = high byte of the value that is printed
X = low byte of the value that is printed
Y = any value

Output

Registers
A = undefined
X = unchanged
Y = unchanged
P = changed

See also

"Standard Output Routines" in Chapter 4

$F944 PrntX

Description This routine prints to standard output the contents of the X register as a two-digit hexadecimal value.

Input

Registers	A = any value
	X = the value that is printed
	Y = any value

Output

Registers	A = undefined
	X = unchanged
	Y = unchanged
	P = changed

See also "The Standard I/O Links" in Chapter 4
"Standard Output Routines" in Chapter 4

$F948 PrBlnk

Description This routine prints three blank spaces to standard output.

Input

Registers	A = any value
	X = any value
	Y = any value

Output

Registers	A = $00
	X = $A0 (the ASCII Space character)
	Y = unchanged
	P = undefined

See also

"The Standard I/O Links" in Chapter 4

"Standard Output Routines" in Chapter 4

$F94A PrBl2

Description

This routine prints to standard output the number of blank spaces specified by the value in the X register. You can print from 1 to 256 spaces with this routine.

Input

Registers
A = any value
X = the number of spaces to print, from $01–$FF; $00 prints 256 spaces
Y = any value

Output

Registers
A = $00
X = $A0 (the ASCII Space character)
Y = unchanged
P = undefined

See also

"The Standard I/O Links" in Chapter 4
"Standard Output Routines" in Chapter 4

$F953 PCAdj

Description

This routine reads from memory the value of the Monitor's program counter, loads the low byte (PCL) into the A register, loads the high byte (PCH) into the Y register, and increments the value in the A and Y registers by 1 to 4. The LENGTH location in zero page (address $2F) contains 1 less than the amount by which PCL/PCH is to be incremented. For example, if LENGTH = 2, then 3 is added to PCL/PCH.

◆ *Note:* The Monitor's program counter (PCL/PCH) in memory is not changed by the PCAdj routine; the incremented value is available only in the A and Y registers at the end of this routine.

Input

Registers
A = any value
X = any value
Y = any value

Memory
PCL/PCH (addresses $3A–$3B) = Monitor's program counter
LENGTH (address $2F) = 1 less than the value to add to PCL/PCH; 0 to 3

Output

Registers
A = new PCL
X = undefined
Y = new PCH
P = undefined

Memory
PCL = unchanged
PCH = unchanged

$FA40 OldIRQ

Description

This routine jumps to the interrupt-handling routine in ROM. The interrupt-handling routine saves the memory state of the machine, checks for an internal interrupt, and then calls the user's interrupt handler at $03FE.

Input

Registers	A = any value
	X = any value
	Y = any value

Output

Registers	A = unchanged
	X = unchanged
	Y = unchanged
	P = unchanged
Memory	A5H (address $45) = value in A register before IRQ interrupt

See also

"User's Interrupt Handler at $03FE" in Chapter 3

$FA47 NewBrk

Description

This routine stores the A register (accumulator) in memory location MACSTAT (address $44) and pulls values for the Y, X, and A registers from the stack before passing control to the Break routine.

◆ *Note:* Normally, NewBrk is called by the interrupt handler, which has set the hardware to its default state and encoded the memory state of the machine in the A register. The encoding scheme for the memory state is listed in the section "Handling Break Instructions" in Chapter 3.

Input

Registers	A = memory state of the machine
	X = any value
	Y = any value

Output

Registers	A = third value from stack
	X = second value from stack
	Y = first value from stack
	P = unchanged
Memory	MACSTAT (address $44) = memory state of the machine

See also

Break
"Handling Break Instructions" in Chapter 3

$FA4C Break

Description

This routine is the 65C02 break handler. Break saves in memory the 65C02 registers and the program counter, and then jumps to the routine pointed to by the user hook at $03F0–$03F1. The default address for this pointer is $FA59, the OldBrk routine.

Input

Registers
- A = any value
- X = any value
- Y = any value

Memory
- A5H (address $45) = the value in the A register before the break

Output

Registers
- A = undefined
- X = undefined
- Y = undefined
- P = undefined

Memory
- A5H (address $45) = the value in the A register when break occurred
- XREG (address $46) = the value in the X register when break occurred
- YREG (address $47) = the value in the Y register when break occurred
- STATUS (address $48) = the value in the P register when break occurred
- SPNT (address $49) = the value in the stack pointer when break occurred
- PCL/PCH (addresses $3A–3B) = address of the program counter when break occurred earlier

See also

OldBrk
"Handling Break Instructions" in Chapter 3

$FA59 OldBrk

Description This routine prints the values for the program counter and registers that are stored in zero page, then jumps to the Monitor.

Input

Registers
A = any value
X = any value
Y = any value

Memory
A5H (address $45) = the value in the A register before the break
XREG (address $46) = the value in the X register before the break
YREG (address $47) = the value in the Y register before the break
STATUS (address $48) = the value in the P register before the break
SPNT (address $49) = the value in the stack pointer before the break
PCL/PCH (addresses $3A–3B) = address of the program counter before the break

Output

Registers
A = undefined
X = undefined
Y = undefined
P = undefined

See also "Handling Break Instructions" in Chapter 3

$FA62 Reset

Description

This routine is called by the 65C02 reset vector at $FFFC–$FFFD. The Reset routine performs a warm start initialization, then checks to see if a cold start is required. If the Command key was pressed during reset or if the user reset vector is not valid, the routine jumps to the cold-start routine, PwrUp. If this reset is a normal warm start, then Reset jumps to the address pointed to by the user interrupt vector at $03F2–$03F3.

Input

Registers	A = any value X = any value Y = any value
Memory	SOFTEV (address 03F2–03F3) = pointer to routine to be executed after a warm start PWREDUP (address 03F4) = Validity-Check byte, used to check SOFTEV for validity

Output

The Reset routine does not return to the calling program.

See also

PwrUp
"Starts and Resets" in Chapter 3

$FAA6 PwrUp

Description

This routine completes the initialization of the machine necessary for a cold start. The PwrUp routine is called by the Reset routine when either the Command key is pressed along with the Control and Reset keys, or the reset vector is not valid (the power-up bytes do not match). PwrUp sets some soft switches, writes 2 bytes of meaningless data into the beginning of each page of main RAM, and passes control to the firmware for the first startup device. If the firmware cannot find any startup routine in any startup device, it displays a message on the screen and passes control to Applesoft.

Input

Registers A = any value
X = any value
Y = any value

Output

The PwrUp routine does not return to the calling program.

See also

Reset
"Starts and Resets" in Chapter 3

$FAD7 RegDsp

Description

This routine displays the memory state of the machine stored in MACSTAT (address $44) plus the A, X, Y, P, and S register contents stored by the firmware. This information is stored by the Break routine before jumping to the routine pointed to by the user hook at $03F0–$03F1. The encoding scheme for the memory state is listed in the section "Handling Break Instructions" in Chapter 3.

Input

Registers
A = any value
X = any value
Y = any value

Memory
MACSTAT (address $44) = memory state of the machine
A5H (address $45) = A register value
XREG (address $46) = X register value
YREG (address $47) = Y register value
STATUS (address $48) = P register value
SPNT (address $49) = stack pointer value

Output

Registers
A = undefined
X = undefined
Y = undefined
P = undefined

See also

"Handling Break Instructions" in Chapter 3

$FB1E PRead

Description

This routine returns a number ($00 to $FF) in the Y register that indicates the position of the dial on a hand control (game paddle). You use the X register to specify which hand control should be read. PRead reads the mouse position in place of the game paddle if a mouse is connected, turned on, and in transparent mode. If X = 0, PRead returns the X position of the mouse in the Y register; if X = 1, PRead returns the Y position of the mouse.

Input

Registers
A = any value
X = hand control to be read; 0 or 1
Y = any value

Output

Registers
A = undefined
X = unchanged
Y = position of hand control dial, $00–$FF
P = undefined

See also

"Game Input" in Chapter 9
"Using the Mouse As a Hand Control" in Chapter 9

$FB21 PRead4

Description

This routine is identical to the PRead routine, except that PRead4 does not read the mouse position if a mouse is connected to the game port.

Input

Registers
A = any value
X = hand control to be read; 0 or 1
Y = any value

Output

Registers
A = undefined
X = unchanged
Y = position of hand control dial, $00–$FF
P = undefined

See also

PRead
"Game Input" in Chapter 9

$FB2F Init

Description

This routine initializes the text screen: it sets the display page to text Page 1 and sets the display to a full-screen text window display. Init also calls BasCalc to calculate the starting address in memory (the base address) of the data for the last line of text on the text screen; BasCalc stores this address in BASL/BASH.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = low byte of base address
X = undefined
Y = undefined
P = undefined

Memory
BASL/BASH (address $28–$29) = base address of last line in window

See also

BasCalc
"Text Modes" in Chapter 6
"Display Page Maps" in Chapter 6

$FB39 SetTxt

Description

This routine sets the display to a full-screen text window display. SetTxt also calls BasCalc to calculate the starting address in memory (the base address) of the data for the last line of text on the text screen; BasCalc stores this address in BASL/BASH. SetTxt is identical to the Init routine, except that SetTxt does not set the display page to text Page 1.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = low byte of base address
X = undefined
Y = undefined
P = undefined

Memory
BASL/BASH (address $28–$29) = base address of last line of text

See also

BasCalc
Init
"Text Modes" in Chapter 6
"Display Page Maps" in Chapter 6

$FB40 SetGr

Description

This routine sets the display to mixed graphics mode, clears the graphics portion of the screen, and sets the top of the text window to line 20. SetGr also calls BasCalc to calculate the starting address in memory of the data for the last line of text on the screen; BasCalc stores this address in BASL/BASH.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = low byte of base address
X = undefined
Y = undefined
P = undefined

Memory
BASL/BASH (address $28–$29) = base address of last line of text

See also

BasCalc
"Mixed-Mode Displays" in Chapter 6

$FB4B SetWnd

Description

This routine sets the text window to the full width of the screen, with the bottom of the window at the bottom of the screen and the top of the window at the value passed in the A register. SetWnd also calls BasCalc to calculate the starting address in memory (the base address) for the last line of text in the window; BasCalc stores this address in BASL/BASH.

Input

Registers
A = top of text window ($00–$17)
X = any value
Y = any value

Output

Registers
A = low byte of base address
X = unchanged
Y = undefined
P = undefined

Memory
WNDLFT (address $20) = $00
WNDWDTH (address $21) = $28 for 40-column display, $50 for 80-column display
WNDTOP (address $22) = value passed in the A register
WNDBTM (address $23) = $18
BASL/BASH (address $28–$29) = base address of last line of text

See also

"The Text Window" in Chapter 4

$FB5B TabV

Description

This routine performs a vertical tab to the line specified by the value in the A register. TabV calls BasCalc to calculate the starting address in memory (the base address) for the text-page line specified by the value in the A register. BasCalc stores this address in BASL/BASH, effectively moving the cursor. TabV differs from VTab and VTabz in that TabV stores the new line number in CV.

Input

	Registers	A = line number ($00–$17)
		X = any value
		Y = any value

Output

	Registers	A = low byte of base address
		X = unchanged
		Y = undefined
		P = undefined
	Memory	CV (address $25) = line number passed in A register
		BASL/BASH (address $28–$29) = base address for specified line

See also

VTab
VTabz
"Standard Output Routines" in Chapter 4
"Text Modes" in Chapter 6
"Display Page Maps" in Chapter 6

$FB60 AppleII

Description

This routine clears the text screen and displays the string "Apple IIc" or "Apple IIc +" on the first line of the screen. This routine does not work when the display is set to graphics or mixed mode.

Input

Registers A = any value
X = any value
Y = any value

Output

Registers A = undefined
X = unchanged
Y = undefined
P = changed

See also

"Text Modes" in Chapter 6

$FB6F SetPwrC

Description This routine calculates the Validity-Check byte for the current reset vector and stores it in memory.

Input

Registers	A = any value X = any value Y = any value	
Memory	SOFTEV + 1 (address $03F3) = the high byte of the reset vector	

Output

Registers	A = the Validity-Check byte X = unchanged Y = unchanged P = undefined	
Memory	PWREDUP (address $03F4) = the Validity-Check byte	

See also "The Reset Vector" in Chapter 3

$FB78 VidWait

Description

This routine checks to see if the A register contains the value $8D (a carriage return); if it does, VidWait checks the keyboard for a Control-S character ($93). If it finds a Control-S, VidWait clears the keyboard strobe and passes control to KbdWait, which causes the output to pause.

If it does not find a Control-S, VidWait checks to see if the enhanced video firmware is active. If enhanced video firmware is not active, VidWait passes control to VidOut, which handles the standard control-character functions, prints the next character, and advances the cursor. If the enhanced video firmware is active, VidWait passes control to a routine that handles the enhanced video control-character functions, prints the next character, and advances the cursor.

VidWait ignores all input from the keyboard other than Control-S.

Input

Registers
A = the next character to be printed
X = any value
Y = any value

Output

Registers
A = undefined
X = unchanged
Y = undefined
P = undefined

See also

"Standard Output Routines" in Chapter 4

$FB88 KbdWait

Description

This routine waits for a keypress. When a key has been pressed, KbdWait checks to see if it is a Control-C. If the character is not a Control-C, KbdWait clears the keyboard strobe and passes control to VidOut, which prints the character in the A register. If the character is a Control C, KbdWait passes control directly to VidOut, without clearing the keyboard strobe.

Input

Registers
A = the next character to be printed
X = any value
Y = any value

Output

Registers
A = undefined
X = unchanged
Y = undefined
P = undefined

See also

"The Stop-List Feature" in Chapter 4

$FBB3 Version

Description

This location contains one of the identification bytes for the ROM. Version is not a callable routine. The value of this byte is $06 for all Apple IIc computers.

Input

This is not a callable routine.

Output

This is not a callable routine.

See also

"Machine Identification" in Appendix D

$FBBF ZIDByte2

Description *This location contains one of the identification bytes for the ROM. ZIDByte2 is not a* callable routine. The value of this byte depends on the version of Apple IIc computer, as shown in Table F-3.

■ **Table F-3** Values of Apple IIc identification byte

Machine	$FBBF
Apple IIc (original)	$FF
Apple IIc (UniDisk 3.5)	$00
Apple IIc (memory expansion)	$03
Apple IIc Plus	$05

Input This is not a callable routine.

Output This is not a callable routine.

See also "Machine Identification" in Appendix D

$FBC0 ZIDByte

Description
This location contains one of the identification bytes for the ROM. ZIDByte is not a callable routine. The value of this byte is $00 for all Apple IIc computers.

Input
This is not a callable routine.

Output
This is not a callable routine.

See also
"Machine Identification" in Appendix D

$FBC1 BasCalc

Description

This routine calculates the starting address in memory (base address) for the line on the text screen specified by the value in the A register. BasCalc stores the address at BASL/BASH.

Input

Registers
A = line number ($00–$17)
X = any value
Y = any value

Output

Registers
A = low byte of base address
X = unchanged
Y = unchanged
P = undefined

Memory
BASL/BASH (address $28–$29) = base address for specified line

See also

"Text Modes" in Chapter 6
"Display Page Maps" in Chapter 6

$FBDD Bell1

Description

This routine clicks the speaker at 1 kHz for 0.1 second to generate a tone. There is a 0.01 second delay before the tone is generated to prevent rapid calls to Bell1 from causing distorted sounds.

Input

Registers A = any value
X = any value
Y = any value

Output

Registers A = undefined
X = unchanged
Y = $00
P = undefined

See also

Bell1.2
Bell2
"Speaker Output" in Chapter 5

$FBE2 Bell1.2

Description

This routine clicks the speaker at 1 kHz for 0.1 second to generate a tone. Bell1.2 differs from Bell1 in that Bell1.2 has no delay before the tone is generated.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = undefined
X = unchanged
Y = $00
P = undefined

See also

Bell1
Bell2
"Speaker Output" in Chapter 5

$FBE4 Bell2

Description

This routine clicks the speaker repeatedly to generate a 1 kHz square-wave tone; each two clicks constitute one cycle. The duration of the tone depends on the value passed in the Y register. A value of $C0 in the Y register toggles the speaker 192 times, generating a tone for approximately 0.1 second. A value of $00 in the Y register toggles the speaker 256 times.

Input

Registers
A = any value
X = any value
Y = number of times to click speaker (duration of tone = 0.001Y/2)

Output

Registers
A = undefined
X = unchanged
Y = $00
P = undefined

See also

Bell1
Bell1.2
"Speaker Output" in Chapter 5

$FBF0 StorAdv

Description

This routine places a printable character on the text screen. The character is placed on the line determined by the value in BASL at the horizontal position determined by the value in CH. After printing the character, StorAdv increments CH to advance the cursor. If CH + 1 exceeds the window width (stored in WNDWDTH), StorAdv executes a carriage return.

Input

Registers	A = character to display X = any value Y = any value
Memory	WNDWDTH (address $21) = window width CH (address $24) = horizontal position of character BASL/BASH (address $28–$29) = base address of current line of text

Output

Registers	A = undefined X = unchanged Y = undefined P = undefined
Memory	CH = previous value of CH + 1

See also

"Standard Output Routines" in Chapter 4
"Text Modes" in Chapter 6
"Display Page Maps" in Chapter 6

$FBF4 Advance

Description

This routine increments the value stored in CH by 1, advancing the cursor. If CH + 1 exceeds the window width (stored in WNDWDTH), Advance executes a carriage return.

Input

Registers	A = any value X = any value Y = any value
Memory	WNDWDTH (address $21) = window width CH (address $24) = horizontal position of cursor

Output

Registers	A = undefined X = unchanged Y = undefined P = undefined
Memory	CH = previous value of CH + 1

See also

"Standard Output Routines" in Chapter 4
"Text Modes" in Chapter 6
"Display Page Maps" in Chapter 6

$FBFD VidOut

Description

This routine sends printable characters to StorAdv. VidOut also checks for carriage returns, line feeds, backspaces, and bell characters (Control-G); if it finds one of these characters, VidOut branches to the appropriate routine to handle it. VidOut ignores all other control characters.

Input

Registers
A = character to display
X = any value
Y = any value

Output

Registers
A = character to display or control character to process
X = unchanged
Y = undefined
P = undefined

See also

StorAdv
"Standard Output Routines" in Chapter 4

$FC10 BS

Description

This routine executes a back space. BS decrements the value in CH by one; if the cursor is at the left edge of the window, CH is set to the right edge of the window and BS branches to the Up routine.

Input

Registers
A = any value
X = any value
Y = any value

Memory
WNDLFT (address $20) = left edge of text window
WNDWDTH (address $21) = text window width
CH (address $24) = horizontal position of cursor

Output

Registers
A = undefined
X = unchanged
Y = undefined
P = undefined

Memory
CH = previous value of CH – 1

See also

"Standard Output Routines" in Chapter 4
"Text Modes" in Chapter 6
"Display Page Maps" in Chapter 6

$FC1A Up

Description

This routine decrements the value CV by 1, moving the cursor up one line, unless the cursor is currently on the top line of the window.

Input

Registers
A = any value
X = any value
Y = any value

Memory
WNDTOP (address $22) = top of window
CV (address $25) = vertical position of cursor

Output

Registers
A = undefined
X = unchanged
Y = unchanged
P = undefined

Memory
CV = previous value of CV – 1 (unless at top of window)

See also

"Standard Output Routines" in Chapter 4
"Text Modes" in Chapter 6
"Display Page Maps" in Chapter 6

$FC22 VTab

Description

This routine performs a vertical tab to the line specified by the value stored in CV. VTab calls BasCalc to calculate the starting address in memory (the base address) for the text-page line specified by the value stored in CV. BasCalc stores this address in BASL/BASH, effectively moving the cursor. VTab differs from TabV in that VTab does not store a new line number in CV.

Input

Registers	A = any value
	X = any value
	Y = any value
Memory	CV (address $25) = line number to which cursor is moved ($00–$17)

Output

Registers	A = low byte of base address
	X = unchanged
	Y = unchanged
	P = undefined
Memory	BASL/BASH (address $28–$29) = base address for specified line

See also

TabV
VTabz
"Standard Output Routines" in Chapter 4
"Text Modes" in Chapter 6
"Display Page Maps" in Chapter 6

$FC24 VTabz

Description

This routine performs a vertical tab to the line specified by the value stored in the A register (accumulator). VTabz calls BasCalc to calculate the starting address in memory (the base address) for the text-page line specified by the value stored in the A register. BasCalc stores this address in BASL/BASH, effectively moving the cursor. VTabz differs from TabV in that it does not store the line number in CV; it differs from VTab in that it does not read the line number from CV.

Input

Registers	A = line number ($00–$17)	
	X = any value	
	Y = any value	

Output

Registers	A = low byte of base address
	X = unchanged
	Y = unchanged
	P = undefined
Memory	BASL/BASH (address $28–$29) = base address for specified line

See also

TabV
VTab
"Standard Output Routines" in Chapter 4
"Text Modes" in Chapter 6
"Display Page Maps" in Chapter 6

$FC42 ClrEOP

Description

This routine clears the text window from the cursor position to end of the current line and from the current line to the bottom of the window, and then returns the cursor to its starting position.

Input

Registers		A = any value X = any value Y = any value
Memory		WNDBTM (address $23) = bottom of window CV (address $25) = vertical position of cursor CH (address 24) = horizontal position of cursor for 40-column screen OURCH (address 057B) = horizontal position of cursor for 80-column screen

Output

Registers	A = undefined X = unchanged Y = undefined P = undefined
Memory	BASL/BASH (address $28–$29) = base address for current line

See also

"Standard Output Routines" in Chapter 4
"Text Modes" in Chapter 6
"Display Page Maps" in Chapter 6

$FC58 Home

Description	This routine moves the cursor to the upper-left corner of the text window and then clears from that position to the bottom of the window.	
Input	Registers	A = any value X = any value Y = any value
	Memory	WNDLFT (address $20) = left edge of window WNDWDTH (address $21) = width of window WNDTOP (address $22) = top of window WNDBTM (address $23) = bottom of window
Output	Registers	A = undefined X = unchanged Y = undefined P = undefined
	Memory	BASL/BASH (address $28–$29) = base address for top line in window
See also	"The Text Window" in Chapter 4	

$FC62 CR

Description

This routine executes a carriage return by moving the cursor to the left edge of the window and then calling the LF routine to move the cursor to the next line on the screen.

Input

Registers	A = any value X = any value Y = any value
Memory	WNDLFT (address $20) = left edge of window

Output

Registers	A = undefined X = unchanged Y = undefined P = undefined

See also

LF
"Standard Output Routines" in Chapter 4

$FC66 LF

Description

This routine increments the value of CV by 1, moving the cursor down one line. If the new vertical position of the cursor is below the bottom of the window, LF calls the Scroll routine to scroll the window and puts the cursor on the bottom line of the window. LF also calls BasCalc to calculate the starting address in memory (the base address) of the data for the new line on the text screen; BasCalc stores this address in BASL/BASH.

Input

Registers	A = any value	
	X = any value	
	Y = any value	
Memory	WNDBTM (address $23) = bottom of window	
	CV (address $25) = vertical position of cursor	

Output

Registers	A = undefined	
	X = unchanged	
	Y = undefined	
	P = undefined	
Memory	CV = previous value of CV + 1 (unless at bottom of window)	
	BASL/BASH (address $28–$29) = base address for new line	

See also

Scroll
"Standard Output Routines" in Chapter 4
"Display Page Maps" in Chapter 6

$FC70 Scroll

Description

This routine scrolls the text window by moving all characters in the window up one line. The cursor remains in the same absolute position in the window. Scroll also calls BasCalc to calculate the starting address in memory (the base address) of the data for the new line on the text screen; BasCalc stores this address in BASL/BASH.

Input

Registers
A = any value
X = any value
Y = any value

Memory
WNDLFT (address $20) = left edge of window
WNDWDTH (address $21) = width of window
WNDTOP (address $22) = top of window
WNDBTM (address $23) = bottom of window
CV (address $25) = vertical position of cursor
CH (address $24) = horizontal position of cursor for 40-column screen
OURCH (address $057B) = horizontal position of cursor for 80-column screen

Output

Registers
A = undefined
X = unchanged
Y = undefined
P = undefined

Memory
BASL/BASH (address $28–$29) = base address for new line

See also

"Standard Output Routines" in Chapter 4
"Display Page Maps" in Chapter 6

$FC9C ClrEOL

Description This routine clears a line of text from the cursor position to the right edge of the window.

Input

Registers
A = any value
X = any value
Y = any value

Memory
WNDLFT (address $20) = left edge of window
WNDWDTH (address $21) = width of window
CH (address $24) = horizontal position of cursor for 40-column screen
OURCH (address $057B) = horizontal position of cursor for 80-column screen

Output

Registers
A = undefined
X = unchanged
Y = undefined
P = undefined

See also

ClrEOLZ
"Standard Output Routines" in Chapter 4
"Display Page Maps" in Chapter 6

$FC9E ClrEOLZ

Description

This routine clears a line of text from the column indicated by the value in the Y register to the right edge of the window. The starting column position is relative to the left edge of the window as stored in WNDLFT; the left edge of the window is column 0.

Input

Registers
A = any value
X = any value
Y = the horizontal position at which to start clearing text

Memory
WNDLFT (address $20) = left edge of window
WNDWDTH (address $21) = width of window

Output

Registers
A = undefined
X = unchanged
Y = undefined
P = undefined

See also

ClrEOL
"Standard Output Routines" in Chapter 4
"Display Page Maps" in Chapter 6

$FCA8 Wait

Description

This routine causes a pause for an amount of time determined by the value in the A register (accumulator).

If the value in the A register is *A*, then the number of cycles of delay caused by the Wait routine in Apple IIc computers is

$0.5(26+27A+5A^2)$

To calculate the minimum time delay (in microseconds) caused by the Wait routine in Apple IIc computers, multiply the number of cycles by 14/14.318181, as follows:

$0.488889(26+27A+5A^2)$ microseconds

The number of cycles of delay caused by the Wait routine in the Apple IIc Plus computer is

$0.5(50+25A+5A^2)+29$

Notice that this routine causes A–41 fewer cycles delay than the Wait routine in Apple IIc computers. The 65C02 may be running at 4 MHz for up to the first 16 cycles of the Wait routine. The minimum time delay (in microseconds) caused by the Wait routine in the Apple IIc Plus computer is therefore $0.977778(0.5(50+25A+5A^2)+13) + 0.25(16)$, or $0.488889(50+25A+5A^2) + 16.711114$ microseconds (minimum).

△ Apple IIc Plus — The Apple IIc Plus continues to run at 1.023 MHz for up to 50 msec after exiting from the Wait routine. △

Input

Registers
A = the value used by the routine to determine the duration of the delay
X = any value
Y = any value

Output	Registers	A = $00
		X = unchanged
		Y = unchanged
		P = undefined

See also "The Apple IIc Plus Accelerator RAM" in Chapter 11

\$FCB4 NxtA4

Description

This routine increments by 1 the 2-byte pointer at A4L/A4H and passes control to NxtA1, which performs a 16-bit comparison of A1L/A1H with A2L/A2H, and then increments A1L/A1H by 1. The NxtA4 routine is called repeatedly by the Move and Verify routines as long as A1L/A1H < A2L/A2H.

Input

Registers
A = any value
X = any value
Y = any value

Memory
A4L/A4H (addresses \$42–\$43) = pointer to be incremented
A1L/A1H (addresses \$3C–\$3D) = pointer to be incremented
A2L/A2H (addresses \$3E–\$3F) = pointer compared with A1L/A1H

Output

Registers
A = undefined
X = unchanged
Y = unchanged
P = changed; C flag is set if A1L/A1H ≥ A2L/A2H

Memory
A4L/A4H = previous value of A4L/A4H + 1
A1L/A1H = previous value of A1L/A1H + 1

See also

NxtA1
Move

$FCBA NxtA1

Description

This routine performs a 16-bit comparison of A1L/A1H with A2L/A2H, and then increments A1L/A1H by 1. NxtA1 is an alternate entry point to the NxtA4 routine; NxtA1 does not increment A4L/A4H, but is otherwise identical to NxtA4.

Input

Registers
A = any value
X = any value
Y = any value

Memory
A1L/A1H (addresses $3C–$3D) = pointer to be incremented
A2L/A2H (addresses $3E–$3F) = pointer compared with A1L/A1H

Output

Registers
A = undefined
X = unchanged
Y = unchanged
P = changed; C flag is set if A1L/A1H ≥ A2L/A2H

Memory
A1L/A1H = previous value of A1L/A1H + 1

See also

NxtA4
Move

$FCC9 HeadR

Description

This address is an obsolete entry point. It does nothing except return to the calling routine with an RTS instruction.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = unchanged
X = unchanged
Y = unchanged
P = unchanged

$FD0C RdKey

Description

This routine loads the A register (accumulator) with the character at the current cursor position and passes control to the FD10 routine, which jumps to an input routine.

Input

Registers
A = any value
X = any value
Y = any value

Memory
CH (address $24) = horizontal position of cursor
BASL/BASH (address $28–$29) = base address of current line

Output

Registers
A = character at current cursor position
X = unchanged
Y = value in CH
P = undefined

See also

FD10
"RdKey Subroutine" in Chapter 4

$FD10 FD10

Description

This routine jumps to the routine whose address is stored in KSWL/KSWH, usually the routine KeyIn (when the enhanced video firmware is not active) or C3KeyIn (when the enhanced video firmware is active). The FD10 address is maintained as an alternative entry point for the KeyIn0 routine; there is no functional difference between these routines.

Input

Registers		A = any value X = any value Y = any value
Memory		KSWL/KSWH (address $38–$39) = address to which to jump

Output

This routine jumps to the routine pointed to by KSWL/KSWH; it does not return to the calling routine.

See also

RdKey
KeyIn
"The Standard I/O Links" in Chapter 4

$FD18 KeyIn0

Description

This routine jumps to the routine whose address is stored in KSWL/KSWH, usually the routine KeyIn (when the enhanced video firmware is not active) or C3KeyIn (when the enhanced video firmware is active).

Input

Registers	A = any value X = any value Y = any value
Memory	KSWL/KSWH (address $38–$39) = address to which to jump

Output

This routine jumps to the routine pointed to by KSWL/KSWH; it does not return to the calling routine.

See also

RdKey
KeyIn
"The Standard I/O Links" in Chapter 4

$FD1B KeyIn

Description

This routine displays a cursor, waits until a key is pressed, then loads the ASCII code for that key into the A register (accumulator), removes the cursor from the display, and returns to the calling program. If Escape mode is set when you call KeyIn, and the user presses the Esc key, then KeyIn processes the Escape sequence. Use the RdChar routine to call KeyIn with Escape mode set.

The cursor displayed by the KeyIn routine is determined by the value stored in CURSOR. If CURSOR = $00, the block cursor normally used by the enhanced video firmware is displayed. If CURSOR = $FF, the checkerboard cursor normally used when the enhanced video firmware is inactive is displayed. If CURSOR is any other value, the inverse of the character represented by that value is displayed.

While the KeyIn routine is waiting for a keypress, it continuously updates RNDL/RNDH, which you can use as a seed for a random-number generator.

The C3KeyIn routine, normally called by RdKey when the enhanced video firmware is active, first calls the COutZ routine to display the solid-block cursor on the screen, and then calls the KeyIn routine.

Input

Registers	A = character at the cursor position	
	X = any value	
	Y = any value	
Memory	CURSOR (address $07FB) = type of cursor to display	

Output	Registers	A = key pressed X = unchanged Y = undefined P = undefined
	Memory	RNDL/RNDH (addresses \$4E–\$4F) = random number

See also RdKey
RdChar
"The KeyIn and C3KeyIn Input Routines" in Chapter 4

$FD35 RdChar

Description This routine sets escape mode to active and then jumps to the RdKey routine. Escape mode is shown in Table F-4.

■ **Table F-4** Escape sequences with RdChar

Escape code	Function
Esc	Clears the window and homes the cursor (places it in the upper-left corner of the screen); exits from escape mode
Esc A or Esc a	Moves the cursor right one line; exits from escape mode
Esc B or Esc b	Moves the cursor left one line; exits from escape mode
Esc C or Esc c	Moves the cursor down one line; exits from escape mode
Esc D or Esc d	Moves the cursor up one line; exits from escape mode
Esc E or Esc e	Clears to the end of the line; exits from escape mode
Esc F or Esc f	Clears to the bottom of the window; exits from escape mode
Esc I or Esc i *or* Esc Up Arrow	Moves the cursor up one line; remains in escape mode
Esc J or Esc j *or* Esc Left Arrow	Moves the cursor left one space; remains in escape mode
Esc K or Esc k *or* Esc Right Arrow	Moves the cursor right one space; remains in escape mode
Esc M or Esc m *or* Esc Down Arrow	Moves the cursor down one line; remains in escape mode
Esc 4	Switches to 40-column mode; sets links to C3KeyIn and C3COut1; restores normal window size (Table 4-5); exits from escape mode*
Esc 8	Switches to 80-column mode; sets links to C3KeyIn and C3COut1; restores normal window size (Table 4-5); exits from escape mode*
Esc Control-D	Disables control characters; only carriage return, linefeed, bell, and backspace have an effect when printed
Esc Control-E	Reactivates control characters
Esc Control-Q	Deactivates the enhanced video firmware; sets links to KeyIn and COut1; restores normal window size (Table 4-5); exits from escape mode*

* This code functions only when the enhanced video firmware is active.

Input	Registers	A = any value X = any value Y = any value
	Memory	CH (address $24) = horizontal position of cursor BASL/BASH (address $28–$29) = base address of current line
Output	Registers	A = character at current cursor position X = unchanged Y = value in CH P = undefined
See also	RdKey "RdKey Subroutine" in Chapter 4	

$FD67 GetLnZ

Description

This routine sends a carriage return to standard output and then passes control to the GetLn routine. The GetLn routine displays the prompt character stored in PROMPT and calls RdKey repeatedly—with escape mode set—to read an entire line of characters from standard input. If the user presses Control-X or backspaces to the first column in the window, GetLn ignores the characters on the line and returns control to GetLnZ.

Input

Registers
A = any value
X = any value
Y = any value

Memory
CH (address $24) = horizontal position of cursor
BASL/BASH (address $28–$29) = base address of current line
PROMPT (address $33) = the ASCII code (high bit set) of the character to be used as a prompt

Output

Registers
A = undefined
X = length of input line
Y = undefined
P = undefined

Memory
INBUF (address $0200–$02xx) = input line

See also

GetLn
RdKey
RdChar
"GetLn Subroutine" in Chapter 4

$FD6A GetLn

Description

This routine displays the prompt character stored in PROMPT and calls RdKey repeatedly—with escape mode set—to read an entire line of characters from standard input. Each character is placed in the input buffer (at $0200) at a relative offset equal to the character's position on the line.

If the user presses the Left Arrow key (Control-H, $88), GetLn moves the cursor left one space, leaves the character on the screen, and places in the input buffer a backspace character ($88) corresponding to the cursor's new position. If the user presses Control-X or backspaces to the first column in the window, GetLn ignores all the characters on the line and jumps to GetLnZ, which executes a carriage return and passes control to GetLn. (If the user presses Control-X, GetLn prints a backslash before jumping to GetLnZ.)

If the user presses the Right Arrow key (Control-U, $95), GetLn reads the character that the cursor was on, and stores it in the input buffer instead of storing $95. For example, suppose you are in the Monitor (which uses GetLn for input) and type the following characters:

300LXY¨¨¨Æ Carriage Return

The input buffer would then contain the following data:

$B3 $B0 $B0 $CC $8D $88

This corresponds to the following string:

300L Carriage Return ¨

Input

Registers	A = any value
	X = any value
	Y = any value

	Memory	CH (address $24) = horizontal position of cursor BASL/BASH (address $28–$29) = base address of current line PROMPT (address $33) = the ASCII code (high bit set) of the character to be used as a prompt if control is passed to GetLnZ
Output	Registers	A = undefined X = length of input line Y = undefined P = undefined
	Memory	INBUF (address $0200–$02xx) = input line
See also	RdKey RdChar "GetLn Subroutine" in Chapter 4	

$FD6C GetLn0

Description

This routine displays as a prompt the character stored in the A register (accumulator), and calls RdKey repeatedly—with escape mode set—to read an entire line of characters from standard input. GetLn0 is an alternate entry point to the GetLn routine. If the user presses Control-X or backspaces to the first column in the window, GetLn ignores the characters on the line and jumps to GetLnZ.

Input

Registers
A = any value
X = any value
Y = any value

Memory
CH (address $24) = horizontal position of cursor
BASL/BASH (address $28–$29) = base address of current line
PROMPT (address $33) = the ASCII code (high bit set) of the character to be used as a prompt if control is passed to GetLnZ

Output

Registers
A = undefined
X = length of input line
Y = undefined
P = undefined

Memory
INBUF (address $0200–$02xx) = input line

See also

GetLn
RdKey
RdChar
"GetLn Subroutine" in Chapter 4

$FD6F GetLn1

Description

This routine calls RdKey repeatedly—with escape mode set—to read an entire line of characters from standard input. GetLn1 is an alternate entry point to the GetLn routine that does not display a prompt character. If the user presses Control-X or backspaces to the first column in the window, GetLn ignores the characters on the line and jumps to GetLnZ.

Input

Registers
A = any value
X = any value
Y = any value

Memory
CH (address $24) = horizontal position of cursor
BASL/BASH (address $28–$29) = base address of current line
PROMPT (address $33) = the ASCII code (high bit set) of the character to be used as a prompt if control is passed to GetLnZ

Output

Registers
A = undefined
X = length of input line
Y = undefined
P = undefined

Memory
INBUF (address $0200–$02xx) = input line

See also

GetLn
RdKey
RdChar
"GetLn Subroutine" in Chapter 4

$FD8B CROut1

Description

This routine clears the current line from the current cursor position to the right edge of the text window, then passes control to CROut, which sends a carriage return to standard output.

Input

Registers A = any value
X = any value
Y = any value

Output

Registers A = carriage return character ($8D)
X = unchanged
Y = undefined
P = undefined

See also

CROut
COut

$FD8E CROut

Description This routine sends a carriage return to standard output.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = carriage return character ($8D)
X = unchanged
Y = undefined
P = undefined

See also

CROut
COut

$FD92 PrA1

Description

This routine sends a carriage return to standard output, then prints to standard output the contents of A1L/A1H in hexadecimal, followed by a hyphen (-). The Verify routine uses the PrA1 routine followed by the PrByte routine to print an address followed by the value in that address during a comparison of memory ranges.

Input

Registers	A = any value X = any value Y = any value
Memory	A1L/A1H (addresses $3C–$3D) = values printed out by this routine

Output

Registers	A = $AD (hyphen) X = undefined Y = $00 P = undefined

See also

PrByte
COut
Verify
"Running a Program" in Chapter 10

$FDDA PrByte

Description

This routine prints to standard output the contents of the A register (accumulator) in hexadecimal format.

Input

Registers
A = value to be printed as a hexadecimal number
X = any value
Y = any value

Output

Registers
A = undefined
X = unchanged
Y = unchanged
P = undefined

See also

PrHex
COut

$FDE3 PrHex

Description

This routine prints to standard output the lower nibble of the contents of the A register (accumulator) in hexadecimal format.

Input

Registers A = value, lower nibble of which is to be printed as a hexadecimal number
X = any value
Y = any value

Output

Registers A = undefined
X = unchanged
Y = unchanged
P = undefined

See also

PrByte
COut

$FDED COut

Description

This routine calls the text output routine whose address is stored in CSWL/CSWH. When the enhanced video firmware is not active, COut normally calls COut1; when the enhanced video firmware is active, COut normally calls C3COut1.

Input

Registers	A = character to be printed X = any value Y = any value
Memory	CSWL/CSWH (addresses $36–$37) = address of output routine to which COut passes control

Output

Registers	A = unchanged X = unchanged Y = unchanged P = undefined

See also

COut1
CoutZ
"The Standard I/O Links" in Chapter 4

$FDF0 COut1

Description

This routine displays the character in the A register (accumulator) on the screen at the current cursor position and advances the cursor. COut1 also handles the following control characters: carriage return ($8D); line feed (Down Arrow, $8C), backspace (Left Arrow, $88); and bell (Control-G, $87). The character displayed depends on the value of the inverse flag INVFLG; COut1 applies the inverse flag unless the character is a control character. See the sections "Primary Character Set Display" and "Alternate Character Set Display" in Chapter 4 for a discussion of the inverse flag.

The C3COut1 routine, normally called by COut when the enhanced video firmware is active, calls the COutZ entry point to the COut1 routine. The enhanced video firmware includes routines to handle a variety of control characters that are ignored when the enhanced video firmware is not active; see the section "Control Characters With C3Cout1" in Chapter 4 for details.

Input

Registers
A = character to be printed
X = any value
Y = any value

Memory
CV (address $25) = vertical position of cursor
CH (address $24) = horizontal position of cursor for 40-column screen
INVFLAG (address $32) = inverse flag
OURCH (address $057B) = horizontal position of cursor for 80-column screen

Output

Registers
A = unchanged
X = unchanged
Y = unchanged
P = undefined

See also

COut
COutZ
"Standard Output Routines" in Chapter 4

$FDF6 COutZ

Description

This address is an alternate entry point to the COut1 routine. The COutZ routine is identical to the COut1 routine, except that the inverse flag is not applied to the character at the beginning of the routine. The enhanced video firmware, which applies the inverse flag after checking for and handling control characters, uses the COutZ entry point to avoid applying the inverse flag twice.

Input

Registers	A = character to be printed
	X = any value
	Y = any value
Memory	CV (address $25) = vertical position of cursor
	CH (address $24) = horizontal position of cursor for 40-column screen
	INVFLAG (address $32) = inverse flag
	OURCH (address $057B) = horizontal position of cursor for 80-column screen

Output

Registers	A = unchanged
	X = unchanged
	Y = unchanged
	P = undefined

See also

COut
COut1
"Standard Output Routines" in Chapter 4

$FE1F IDRoutine

Description

This routine, when called with the c flag set, returns with the c flag still set. If the c flag were returned clear, that would indicate that the machine is an Apple IIGS or later system, and the A, X, and Y registers would contain identification information.

Input

Registers
A = any value
X = any value
Y = any value
P = any value, except that c flag must be set

Output

Registers
A = unchanged
X = unchanged
Y = unchanged
P = unchanged

$FE2C Move

Description

This routine copies the contents of memory from one range of locations into another range of locations. The Move routine reads the contents of memory pointed to by the address in A1L/A1H, plus any offset in the Y register; the instruction is `LDA (A1L),Y`. Then the routine stores the data in the location pointed to by A4L/A4H; the instruction is `STA (A4L),Y`. The address of the end of the source buffer is stored in A2L/A2H.

The Move routine calls the NxtA4 routine. The NxtA4 routine increments A4L/A4H and A1L/A1H and compares A1L/A1H to A2L/A2H, then returns to the Move routine with an RTS instruction. If A1L/A1H < A2L/A2H (the c flag is clear), the Move routine loops back to the Move entry point; if A1L/A1H ≥ A2L/A2H, the Move routine exits with an RTS instruction.

Input

Registers
A = any value
X = any value
Y = initial offset into source and destination buffers; normally = $00

Memory
A1L/A1H (addresses $3C–$3D) = pointer to start of source buffer
A2L/A2H (addresses $3E–$3F) = pointer to end of source buffer
A4L/A4H (addresses $42–$43) = pointer to start of destination buffer

Output

Registers
A = undefined
X = unchanged
Y = unchanged
P = undefined

Memory
A1L/A1H = pointer to end of source buffer + 1
A2L/A2H = pointer to end of source buffer
A4L/A4H = pointer to end of destination buffer + 1

See also

"Moving Data in Memory" in Chapter 10

$FE36 Verify

Description

This routine compares the contents of two ranges of memory. The Verify routine reads the contents of memory pointed to by the address in A1L /A1H, plus any offset in the Y register; the instruction is `LDA (A1L),Y`. Then the routine reads the data in the location pointed to by A4L/A4H; the instruction is `CMP (A4L),Y`. The final address to be compared is stored in A2L/A2H. If the contents of one address in the first block of data do not match those of the corresponding address in the second block of data, Verify prints out the first address, a hyphen (-), the contents of the address in the first block, and (in parentheses) the contents of the address in the second block.

For example, if you call Verify with $0300–$030F in A1L/A1H–A2L/A2H and $0350 in A4L/A4H, and if the value in $0305 is $1F while the value in $0355 is $AE, Verify prints out the following line:

```
0305-1F (AE)
```

Input

Registers	A = any value	
	X = any value	
	Y = initial offset into blocks of data; normally = $00	
Memory	A1L/A1H (addresses $3C–$3D) = pointer to start of first block of data	
	A2L/A2H (addresses $3E–$3F) = pointer to end of first block of data	
	A4L/A4H (addresses $42–$43) = pointer to start of second block of data	

Output

Registers	A = undefined	
	X = unchanged	
	Y = unchanged	
	P = undefined	
Memory	A1L/A1H = pointer to end of first block of data + 1	
	A2L/A2H = pointer to end of first block of data	
	A4L/A4H = pointer to end of second block of data + 1	

See also

"Comparing Data in Memory" in Chapter 10

$FE5E List

Description

This routine disassembles 20 instructions, starting at the address in A1L/A1H, and prints them to standard output.

Input

Registers	A = any value X = $01 Y = any value
Memory	A1L/A1H (addresses $3C–$3D) = pointer to first address to disassemble

Output

Registers	A = undefined X = undefined Y = undefined P = undefined

See also

"Disassembled Programs" in Chapter 10

$FE80 SetInv

Description

This routine sets INVFLG to $3F so that subsequent text that is output through COut1 will be displayed as inverse characters.

Input

Registers	A = any value X = any value Y = any value

Output

Registers	A = unchanged X = unchanged Y = undefined P = undefined
Memory	INVFLG (address $32) = $3F

See also

COut1
SetNorm
"Normal, Inverse, and Flashing Text" in Chapter 4

$FE84 SetNorm

Description

This routine sets INVFLG to $FF so that subsequent text that is output through COut1 will be displayed as normal characters.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = unchanged
X = unchanged
Y = undefined
P = undefined

Memory
INVFLG (address $32) = $FF

See also

COut1
SetInv
"Normal, Inverse, and Flashing Text" in Chapter 4

$FE89 SetKbd

Description

This routine sets the input links KSWL/KSWH to point to the keyboard input routine, KeyIn.

Input

Registers	A = any value
	X = any value
	Y = any value

Output

Registers	A = undefined
	X = undefined
	Y = undefined
	P = undefined
Memory	KSWL/KSWH (addresses $38–$39) = KeyIn ($FD1B)

See also

KeyIn
InPort
"The Standard I/O Links" in Chapter 4

$FE8B InPort

Description

This routine sets the input links KSWL/KSWH to point to the ROM code for the port whose number is in the A register (accumulator). The next attempt to get input through the standard input links will then execute the ROM code that starts at the entry point for the specified port. Setting the A register to $00 and calling InPort is equivalent to calling the SetKbd routine.

Input

Registers
A = number of port ($00–$07)
X = any value
Y = any value

Output

Registers
A = undefined
X = undefined
Y = undefined
P = undefined

Memory
KSWL/KSWH (addresses $38–$39) = $Cn00, where n is the number in the A register

See also

SetKbd
"The Standard I/O Links" in Chapter 4
"Standard Link Entry Points" in Chapter 4

$FE93 SetVid

Description

This routine sets the output links CSWL/CSWH to point to the screen display routine COut1.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = undefined
X = undefined
Y = undefined
P = undefined

Memory
CSWL/CSWH (addresses $36–$37) = COut1 ($FDF0)

See also

COut1
OutPort
"The Standard I/O Links" in Chapter 4

$FE95 OutPort

Description

This routine sets the output hooks CSWL/CSWH to point to the ROM code for the port whose number is in the A register (accumulator). The next attempt to send output through the standard output hooks will then execute the ROM code that starts at the entry point for the specified port. Setting the A register to $00 and calling OutPort is equivalent to calling the SetVid routine.

Input

Registers	A = number of port ($00–$07) X = any value Y = any value

Output

Registers	A = undefined X = undefined Y = undefined P = undefined
Memory	CSWL/CSWH (addresses $36–$37) = $Cn00, where n is the number in the A register

See also

SetVid
"The Standard I/O Links" in Chapter 4
"Standard Link Entry Points" in Chapter 4

$FEB6 Go

Description

This routine sets the A, X, Y, and P registers to the values stored in A5H, XREG, YREG, and STATUS, then jumps to the address stored in A1L/A1H.

Input

Registers
A = any value
X = $01
Y = any value

Memory
A1L/A1H (addresses $3C–$3D) = the starting address of the routine to be executed
A5H (address $45) = the value to which the A register is set
XREG (address $46) = the value to which the X register is set
YREG (address $47) = the value to which the Y register is set
STATUS (address $48) = the value to which the P register is set

Output

Registers
A = undefined
X = undefined
Y = undefined
P = undefined

See also

"Running a Program" in Chapter 10

$FECD Write

Description

This address is an obsolete entry point. It does nothing except return to the calling routine with an RTS instruction.

Input

Registers	A = any value
	X = any value
	Y = any value

Output

Registers	A = unchanged
	X = unchanged
	Y = unchanged
	P = unchanged

$FEFD Read

Description This address is an obsolete entry point. It does nothing except return to the calling routine with an RTS instruction.

Input Registers A = any value
X = any value
Y = any value

Output Registers A = unchanged
X = unchanged
Y = unchanged
P = unchanged

$FF2D PrErr

Description

This routine prints the letters ERR to standard output and sends a bell character ($87) to standard output.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = $87 (bell character)
X = unchanged
Y = unchanged
P = undefined

$FF3A Bell

Description

This routine sends a bell character ($87) to standard output.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = $87 (bell character)
X = unchanged
Y = unchanged
P = undefined

See also

Bell1
Bell1.2
Bell2

$FF3F Restore

Description

This routine sets the A, X, Y, and P registers to the values stored in A5H, XREG, YREG, and STATUS.

Input

Registers
A = any value
X = any value
Y = any value

Memory
A5H (address $45) = the value to which the A register is to be set
XREG (address $46) = the value to which the X register is to be set
YREG (address $47) = the value to which the Y register is to be set
STATUS (address $48) = the value to which the P register is to be set

Output

Registers
A = the value in A5H
X = the value in XREG
Y = the value in YREG
P = the value in STATUS

See also

Save
"Monitor Register Commands" in Chapter 10

$FF4A Save

Description

This routine stores the contents of the A, X, Y, P, and S registers in A5H, XREG, YREG, STATUS, and SPNT. At the end of the routine, Save clears the 65C02 processor's decimal mode flag.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = undefined
X = undefined
Y = unchanged
P = changed; decimal mode flag cleared

Memory
A5H (address $45) = the value in the A register when Save was called
XREG (address $46) = the value in the X register when Save was called
YREG (address $47) = the value in the Y register when Save was called
STATUS (address $48) = the value in the P register when Save was called
SPNT (address ($49) = 2 less than the value in the S register when Save was called

See also

Restore
"Monitor Register Commands" in Chapter 10

$FF58 IORTS

Description

This address contains an RTS instruction. In Apple II machines with expansion slots, a peripheral card can use this RTS instruction to determine which slot it is in.

Input

Registers
A = any value
X = any value
Y = any value

Output

Registers
A = unchanged
X = unchanged
Y = unchanged
P = unchanged

$FF59 OldRst

Description

This routine sets the text display to normal characters, initializes the text screen, sets the output hooks CSWL/CSWH to point to the screen display routine, COut1, and sets the input hooks KSWL/KSWH to point to the keyboard input routine, KeyIn. The OldRst routine then passes control to Mon, which clears the 65C02 processor's decimal mode flag, sounds the speaker, and enters the Monitor.

Input

Registers A = any value
X = any value
Y = any value

Output

The OldRst routine does not return to the calling program.

Memory INVFLG (address $32) = $FF
BASL/BASH (address $28–$29) = base address of last line in window
CSWL/CSWH (addresses $36–$37) = COut1 ($FDF0)
KSWL/KSWH (addresses $38–$39) = KeyIn ($FD1B)

See also

SetNorm
Init
SetVid
SetKbd
Mon

$FF65 Mon

Description This routine clears the 65C02 processor's decimal mode flag, sounds the speaker, and enters the Monitor at MonZ.

Input Registers A = any value
X = any value
Y = any value

Output The Mon routine does not return to the calling program.

See also OldRst
MonZ

$FF69 MonZ

Description

This address is the entry point for the System Monitor to which control is passed when you execute the CALL -151 command from Applesoft. The MonZ routine calls the GetLnZ routine to display the Monitor's asterisk (*) prompt and read a line of text, calls ZMode to clear the Monitor mode, and passes control to the Monitor's command-line parser.

Input

Registers A = any value
X = any value
Y = any value

Output

The MonZ routine does not return to the calling program.

See also

GetLnZ
OldRst
Mon
ZMode
"Invoking the Monitor" in Chapter 10

$FF70 MonZ4

Description

This address is an alternate entry point to the System Monitor. It is the same as the MonZ routine, except that it does not call GetLnZ or ZMode. Your program must read a line of text and clear the Monitor mode before calling MonZ4.

Input

Registers A = any value
X = any value
Y = any value

Output

The MonZ4 routine does not return to the calling program.

See also

GetLnZ
MonZ
ZMode
"Invoking the Monitor" in Chapter 10

$FF8A Dig

Description

This routine converts an ASCII code for a hexadecimal digit in the Monitor's input buffer into the appropriate hexadecimal number and stores that number in A2L/A2H. Dig is called by NxtChr, which performs an exclusive OR operation on each ASCII code with $B0 before passing it to Dig in the A register (accumulator). If the character is from $A to $F, NxtChr also adds $88 to the result of the EOR operation and sets the carry flag before calling Dig. Dig shifts the character bit-by-bit into A2L/A2H in memory. This combination of the NxtChr routine and the Dig routine converts the ASCII representation of a digit into the correct number in memory; for example, the ASCII code $B3 ends up in A2L/A2H as $00 $03. Dig then passes control back to NxtChr.

Input

Registers	A = ASCII code after conditioning by NxtChr routine X = any value Y = offset into input buffer for next character to decode	
Memory	$0200 = start of input buffer	

Output

Registers	A = undefined X = undefined Y = offset into input buffer for next character to decode P = undefined	
Memory	A2L/A2H (addresses $3E–$3F) = hexadecimal number decoded from ASCII character in input buffer	

See also

NxtChr

$FFA7 GetNum

Description

This routine scans the Monitor's input buffer starting at the offset in the Y register. It decodes ASCII codes for hexadecimal numbers into their corresponding hexadecimal values and stores them in A2L/A2H until it encounters an ASCII code that doesn't correspond to a hexadecimal number. GetNum uses NxtChr to test, parse, and decode the hexadecimal numbers.

Input

Registers	A = any value X = any value Y = offset into input buffer for first character to decode
Memory	$0200 = start of input buffer

Output

Registers	A = undefined X = undefined Y = offset into input buffer for next character to decode P = undefined
Memory	A2L/A2H (addresses $3E–$3F) = hexadecimal number decoded from ASCII character in input buffer

See also

NxtChr

$FFAD NxtChr

Description

This routine tests each character in the input buffer to see if it is an ASCII code for a hexadecimal number. If the code is for a hexadecimal number, NxtChr calls Dig to decode the ASCII value into its corresponding hexadecimal number and stores the number in A2L/A2H. NxtChr also converts any lowercase ASCII values into their uppercase equivalents.

Input

Registers
A = any value
X = any value
Y = offset into input buffer for first character to decode

Memory
$0200 = start of input buffer

Output

Registers
A = undefined
X = undefined
Y = offset into input buffer for next character to decode
P = undefined

Memory
A2L/A2H (addresses $3E–$3F) = hexadecimal number decoded from ASCII character in input buffer

See also

GetNum
Dig

$FFC7 ZMode

Description

This routine stores $00 in the Monitor's Mode byte to clear the Monitor mode. The Mode byte is used by the Monitor to determine how to handle hexadecimal numbers that are included in the input line, as shown in Table F-5.

■ **Table F-5** Monitor modes

MODE	Meaning	Command-line symbol
$AB	Addition mode	+
$AD	Subtraction mode	–
$AE	Memory display (examine) mode	.
$BA	Memory fill (data store) mode	:

Input

Registers A = any value
X = any value
Y = any value

Output

Registers A = unchanged
X = unchanged
Y = $00
P = undefined

Memory MODE (address $31) = $00

See also

ToSub
"Monitor Memory Commands" in Chapter 10
"Hexadecimal Arithmetic" in Chapter 10

Appendix G **Conversion Tables**

This appendix briefly discusses bits and bytes and what they can represent, and then discusses peripheral identification numbers. It also contains conversion tables for hexadecimal to decimal and negative decimal, and a number of 8-bit codes.

These tables are intended for convenient reference. This appendix is not intended as a tutorial for the materials discussed. The brief section introductions are for orientation only. ■

Bits and bytes

This section discusses the relationships between bit values and their position within a byte. Here are some rules of thumb regarding the 65C02:

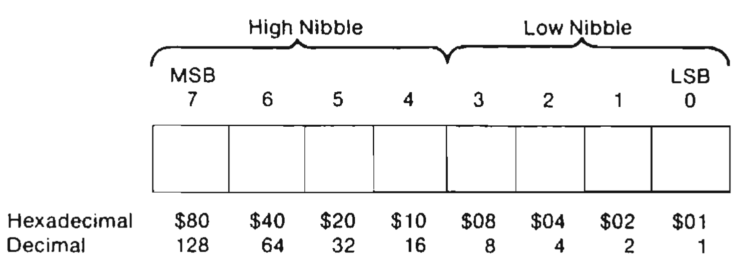
- A bit is a binary digit; it can be either a 0 or a 1.
- A bit can be used to represent any two-way choice. Some choices that a bit can represent in the Apple IIc–family computers are listed in Table G-1.
- Bits can also be combined in groups of any size to represent numbers. Most of the commonly used sizes are multiples of 4 bits.
- Four bits make a nibble (sometimes spelled *nybble*).
- One nibble can represent any of 16 values. Each of these values is assigned a number from 0 through 9 and (because our decimal system has only 10 of the 16 digits we need) A through F.
- Eight bits (two nibbles) make a byte (see Figure G-1).
- One byte can represent any of 16 x 16 (or 256) values. The value can be specified by exactly two hexadecimal digits.
- Bits within a byte are numbered from bit 0 on the right to bit 7 on the left.
- The bit number is the same as the power of 2 that it represents, in a manner completely analogous to the digits in a decimal number.
- One memory position in the Apple IIc–family computers contains one 8-bit byte of data.
- How byte values are interpreted depends on whether the byte is an instruction in a language, part or all of an address, an ASCII code, or some other form of data. Tables G-6 through G-9 list some of the ways bytes are commonly interpreted.
- Two bytes make a word. The 16 bits of a word can represent any one of 256 x 256 (or 65,536) different values.
- The 65C02 uses a 16-bit word to represent memory locations. It can therefore distinguish among 65,536 (64 KB) locations at any given time.
- A memory location is one byte of a 256-byte page. The low-order byte of an address specifies this byte. The high-order byte specifies the memory page the byte is on.

■ **Table G-1** What a bit can represent

Context	Representing	0 =	1 =
Binary number	Place value	0	1 x that power of 2
Logic	Condition	False	True
Any switch	Position	Off	On
Any switch	Position	Clear*	Set
Serial transfer	Beginning	Start	Carrier (no information yet)
Serial transfer	Data	0 value	1 value
Serial transfer	Parity	SPACE	MARK
Serial transfer	End		Stop bit(s)
Serial transfer	Communication state	BREAK	Carrier
P reg. bit n	Neg. result?	No	Yes
P reg. bit v	Overflow?	No	Yes
P reg. bit b	BRK command?	No	Yes
P reg. bit d	Decimal mode?	No	Yes
P reg. bit i	IRQ interrupts	Enabled	Disabled (masked out)
P reg. bit z	Zero result?	No	Yes
P reg. bit c	Carry required?	No	Yes

* Sometimes ambiguously termed reset.

■ **Figure G-1** Bits, nibbles, and bytes



■ **Table G-2** Values represented by a nibble

Binary	Hex	Dec	Binary	Hex	Dec
0000	$0	0	1000	$8	8
0001	$1	1	1001	$9	9
0010	$2	2	1010	$A	10
0011	$3	3	1011	$B	11
0100	$4	4	1100	$C	12
0101	$5	5	1101	$D	13
0110	$6	6	1110	$E	14
0111	$7	7	1111	$F	15

Hexadecimal and decimal

Use Table G-3 for conversion of hexadecimal and decimal numbers.

■ **Table G-3** Hexadecimal/decimal conversion

Digit	$x000	$0x00	$00x0	$000x
F	61440	3840	240	15
E	57344	3584	224	14
D	53248	3328	208	13
C	49152	3072	192	12
B	45056	2816	176	11
A	40960	2560	160	10
9	36864	2304	144	9
8	32768	2048	128	8
7	28672	1792	112	7
6	24576	1536	96	6
5	20480	1280	80	5
4	16384	1024	64	4
3	12288	768	48	3
2	8192	512	32	2
1	4096	256	16	1

To convert a hexadecimal number to a decimal number, find the decimal numbers corresponding to the positions of each hexadecimal digit. Write them down and add them up.

For example:

```
$3C =  ?         $ FD47  =      ?

$30 = 48         $ F000  = 61440
$0C = 12         $  D00  =  3328
$3C = 60         $   40  =    64
                 $    7  =     7
                 $ FD47  = 64839
```

To convert a decimal number to hexadecimal, subtract from the decimal number the largest decimal entry in the table that is less than it. Write down the hexadecimal digit (noting its place value) also. Now subtract the largest decimal number in the table that is less than the decimal remainder, and write down the next hexadecimal digit. Continue until you have a remainder less than 16. Add up the hexadecimal numbers.

For example:

```
16215        =     $     ?

16215 -12288 = 3927        12288 = $ 3000
 3927 - 3840 =   87         3840 = $  F00
   87 -   80 =    7           80 = $   50
    7                          7 = $    7
                           16215 = $ 7F57
```

Hexadecimal and negative decimal

If a number is larger than decimal 32,767, Applesoft BASIC allows and Integer BASIC requires you to use the negative decimal equivalent of the number. Table G-4 is set up to make it easy for you to convert a hexadecimal number directly to a negative decimal number.

■ **Table G-4** Hexadecimal to negative decimal conversion

Digit	$x000	$0x00	$00x0	$000x
F	0	0	0	-1
E	-4096	-256	-16	-2
D	-8192	-512	-32	-3
C	-12288	-768	-48	-4
B	-16384	-1024	-64	-5
A	-20480	-1280	-80	-6
9	-24576	-1536	-96	-7
8	-28672	-1792	-112	-8
7		-2048	-128	-9
6		-2304	-144	-10
5		-2560	-160	-11
4		-2816	-176	-12
3		-3072	-192	-13
2		-3328	-208	-14
1		-3584	-224	-15
0		-3840	-240	-16

To perform this conversion, write down the four decimal numbers corresponding to the four hexadecimal digits (0's included). Then add their values (ignoring their signs for a moment). The resulting number, with a minus sign in front of it, is the desired negative decimal number.

For example:

```
$C010  =      ?
$C000  = -12288
$ 000  =  -3840
$  10  =   -224
$   0  =    -16
-----------------
$C010  = -16368
```

To convert a negative decimal number directly to a positive decimal number, add it to 65,536. (This addition ends up looking like subtraction.)

For example:

-151 = + ?
65536 + (-151) = 65536 - 151 = 65385

To convert a negative decimal number to a hexadecimal number, first convert it to a positive decimal number, then use Table G-3.

Peripheral identification numbers

Many Apple products now use peripheral identification numbers (called *PINs*) as shorthand to designate serial device characteristics. The Apple II–series *Universal Utilities* disk presents a menu from which to select the characteristics of, say, a printer or modem. From the selections made, it generates a PIN for the user. Other products have a ready-made PIN that the user can simply type in.

Table G-5 is a definition of the PIN digits. When communication mode is selected, the seventh digit is ignored.

For example, a PIN of 252/111 means that a device operates with the following characteristics:

Communications mode; 8 data bits, 1 stop bit; 300 baud/no parity; no echo of output to display; no LF after CR; no CRs generated

■ **Table G-5** PIN digits

---

x x x x x x x

1 = Printer mode
2 = Communication mode*

1 = 6 data bits, 1 stop bit
2 = 6 data bits, 2 stop bits
3 = 7 data bits, 1 stop bit
4 = 7 data bits, 2 stop bits
5 = 8 data bits, 1 stop bit
6 = 8 data bits, 2 stop bits

1 = 110 bits per second
2 = 300 bits per second
3 = 1200 bits per second
4 = 2400 bits per second
5 = 4800 bits per second
6 = 9600 bits per second
7 = 19200 bits per second

---

1 = No parity
2 = Even parity (total on = even)
3 = Odd parity (total on = odd)
4 = MARK parity (parity bit = 1)
5 = SPACE parity (parity bit = 0)

1 = Do not echo output on screen
2 = Echo output on screen

1 = Do not generate LF after CR
2 = Generate LF after CR

1 = Do not generate CR*
2 = Generate CR after 40 characters
3 = Generate CR after 72 characters
4 = Generate CR after 80 characters
5 = Generate CR after 132 characters

* If you select communication mode, then the seventh digit must equal 1. This value is supplied automatically when you use the UUD.

Apple IIc–family character set

Tables G-6 through G-9 show the entire ASCII character set. Note that character values are shown with the high bit off. Unless otherwise noted, all ASCII character values above $7F (127 decimal) generate the same character as that value with the high bit off. Here is how to interpret these tables:

- The *Binary* column has the 8-bit code for each ASCII character.
- The first 128 ASCII entries represent 7-bit ASCII codes plus a high-order bit of 0—for example, 01001000 for the letter H.
- A transmitted or received ASCII character will take whichever form (in the communication register) is appropriate if odd or even parity is selected—for example, 11001000 for an odd-parity H, 01001000 for an even-parity H.
- The *ASCII char* column gives the ASCII character name.
- The *Interpretation* column spells out the meaning of special symbols and abbreviations, where necessary.
- The *What to type* column indicates what keystrokes generate the ASCII character (where it is not obvious).
- The columns marked *Pri* and *Alt* indicate what displayed character results from each code when using the primary or alternate display character set, respectively. Boldface is used for inverse characters; italic is used for flashing characters.

  Note that the values $40 through $5F (and $C0 through $DF) in the alternate character set are displayed as MouseText characters (Figure 6-2) if the firmware is set to do so, or if the firmware is bypassed.

◆ *Note:* The primary character set is normally displayed when the enhanced video firmware is inactive, and the alternate character set is normally displayed when the enhanced video firmware is active. The AltChar soft switch shown in Table B-5 can be used to control which character set is displayed.

■ **Table G-6** Control characters

Binary	Dec	Hex	ASCII char	Interpretation	What to type	Pri	Alt
0000000	0	$00	NUL	Blank (null)	Control-@	@	@
0000001	1	$01	SOH	Start of header	Control-A	A	A
0000010	2	$02	STX	Start of text	Control-B	B	B
0000011	3	$03	ETX	End of text	Control-C	C	C
0000100	4	$04	EOT	End of transm.	Control-D	D	D
0000101	5	$05	ENQ	Enquiry	Control-E	E	E
0000110	6	$06	ACK	Acknowledge	Control-F	F	F
0000111	7	$07	BEL	Bell	Control-G	G	G
0001000	8	$08	BS	Backspace	Control-H or Left Arrow-H	H	H
0001001	9	$09	HT	Horizontal tab	Control-I or Tab	I	I
0001010	10	$0A	LF	Line feed	Control-J or Down Arrow-J	J	J
0001011	11	$0B	VT	Vertical tab	Control-K or Up Arrow	K	K
0001100	12	$0C	FF	Form feed	Control-L	L	L
0001101	13	$0D	CR	Carriage return	Control-M or Return	M	M
0001110	14	$0E	SO	Shift out	Control-N	N	N
0001111	15	$0F	SI	Shift in	Control-O	O	O
0010000	16	$10	DLE	Data link escape	Control-P	P	P
0010001	17	$11	DC1	Device control 1	Control-Q	Q	Q
0010010	18	$12	DC2	Device control 2	Control-R	R	R
0010011	19	$13	DC3	Device control 3	Control-S	S	S
0010100	20	$14	DC4	Device control 4	Control-T	T	T
0010101	21	$15	NAK	Neg. acknowledge	Control-U or Right Arrow	U	U
0010110	22	$16	SYN	Synchronization	Control-V	V	V
0010111	23	$17	ETB	End of text block	Control-W	W	W
0011000	24	$18	CAN	Cancel	Control-X	X	X
0011001	25	$19	EM	End of medium	Control-Y	Y	Y
0011010	26	$1A	SUB	Substitute	Control-Z	Z	Z
0011011	27	$1B	ESC	Escape	Control-[or Esc	[	[
0011100	28	$1C	FS	File separator	Control-\	\	\
0011101	29	$1D	GS	Group separator	Control-]	]	]
0011110	30	$1E	RS	Record separator	Control-^	^	^
0011111	31	$1F	US	Unit separator	Control-_	_	_

■ **Table G-7** Special characters

Binary	Dec	Hex	ASCII char	Interpretation	What to type	Pri	Alt
0100000	32	$20	SP	Space	Space bar		
0100001	33	$21	!			!	!
0100010	34	$22	"			"	"
0100011	35	$23	#			#	#
0100100	36	$24	$			$	$
0100101	37	$25	%			%	%
0100110	38	$26	&			&	&
0100111	39	$27	'	Apostrophe		'	'
0101000	40	$28	(			(	(
0101001	41	$29	)			)	)
0101010	42	$2A	*			*	*
0101011	43	$2B	+			+	+
0101100	44	$2C	,	Comma		,	,
0101101	45	$2D	-	Hyphen		-	-
0101110	46	$2E	.	Period		.	.
0101111	47	$2F	/			/	/
0110000	48	$30	0			0	0
0110001	49	$31	1			1	1
0110010	50	$32	2			2	2
0110011	51	$33	3			3	3
0110100	52	$34	4			4	4
0110101	53	$35	5			5	5
0110110	54	$36	6			6	6
0110111	55	$37	7			7	7
0111000	56	$38	8			8	8
0111001	57	$39	9			9	9
0111010	58	$3A	:			:	:
0111011	59	$3B	;			;	;
0111100	60	$3C	<			<	<
0111101	61	$3D	=			=	=
0111110	62	$3E	>			>	>
0111111	63	$3F	?			?	?

■ **Table G-8** Uppercase characters

Binary	Dec	Hex	ASCII char	Interpretation	What to type	Pri	Alt
1000000	64	$40	@			*@*	🍎
1000001	65	$41	A			*A*	🍏
1000010	66	$42	B			*B*	↖
1000011	67	$43	C			*C*	⌛
1000100	68	$44	D			*D*	✓
1000101	69	$45	E			*E*	☑
1000110	70	$46	F			*F*	🏃
1000111	71	$47	G			*G*	🏃
1001000	72	$48	H			*H*	←
1001001	73	$49	I			*I*	…
1001010	74	$4A	J			*J*	↓
1001011	75	$4B	K			*K*	↑
1001100	76	$4C	L			*L*	‾
1001101	77	$4D	M			*M*	↵
1001110	78	$4E	N			*N*	■
1001111	79	$4F	O			*O*	⇠
1010000	80	$50	P			*P*	⇢
1010001	81	$51	Q			*Q*	⇣
1010010	82	$52	R			*R*	⇡
1010011	83	$53	S			*S*	—
1010100	84	$54	T			*T*	⌊
1010101	85	$55	U			*U*	→
1010110	86	$56	V			*V*	▒
1010111	87	$57	W			*W*	▒
1011000	88	$58	X			*X*	⊏
1011001	89	$59	Y			*Y*	⊐
1011010	90	$5A	Z			*Z*	▕
1011011	91	$5B	[	Opening Bracket		*[*	◆
1011100	92	$5C	\	Reverse Slant		**	═
1011101	93	$5D	]	Closing Bracket		*]*	╬
1011110	94	$5E	^	Caret		^	⊡
1011111	95	$5F	_	Underline		_	▏

• If the high bit is set, the MouseText characters are replaced with their equivalent in the primary character set with that value.

■ **Table G-9** Lowercase characters

Binary	Dec	Hex	ASCII char	Interpretation	What to type	Pri	Alt
1100000	96	$60	`	Grave accent			`
1100001	97	$61	a			!	a
1100010	98	$62	b			"	b
1100011	99	$63	c			#	c
1100100	100	$64	d			$	d
1100101	101	$65	e			%	e
1100110	102	$66	f			&	f
1100111	103	$67	g			'	g
1101000	104	$68	h			(	h
1101001	105	$69	i			)	i
1101010	106	$6A	j			*	j
1101011	107	$6B	k			+	k
1101100	108	$6C	l			,	l
1101101	109	$6D	m			-	m
1101110	110	$6E	n			.	n
1101111	111	$6F	o			/	o
1110000	112	$70	p			0	p
1110001	113	$71	q			1	q
1110010	114	$72	r			2	r
1110011	115	$73	s			3	s
1110100	116	$74	t			4	t
1110101	117	$75	u			5	u
1110110	118	$76	v			6	v
1110111	119	$77	w			7	w
1111000	120	$78	x			8	x
1111001	121	$79	y			9	y
1111010	122	$7A	z			:	z
1111011	123	$7B	{	Opening brace		;	{
1111100	124	$7C	|	Vertical line		<	|
1111101	125	$7D	}	Closing brace		=	}
1111110	126	$7E	~	Overline (tilde)		>	~
1111111	127	$7F	DEL	Delete/rubout		?	DEL

Appendix H **Controlling the Apple IIc Plus Accelerator**

This appendix describes how an assembly-language program can control the cache glue gate array (CGGA) chip in the Apple IIc Plus and provides code samples that you can incorporate into your program. This information is provided for the sake of completeness only. Any code that changes speed settings for ports or that disables the CGGA cannot be run on any machine other than the Apple IIc Plus. Any code that does not strictly adhere to the guidelines in this appendix is guaranteed not to work on future versions of the Apple IIc Plus. See the section "The Apple IIc Plus Cache Glue Gate Array (CGGA)" in Chapter 11 for a general description of the CGGA and for approved methods of controlling the speed of the Apple IIc Plus. ■

When the Apple IIc Plus is switched on or reset, ROM code for ports 1, 2, 5, and 6, and the code for the speaker and game paddles cannot be cached; this code is restricted to running at 1.023 MHz. The code for ports 3, 4, and 7 can be cached, and so can run at up to 4 MHz. These settings were chosen to allow code written for other Apple II computers to run on the Apple IIc Plus; we recommend that you never change these settings. Before you decide whether or not to change these default settings or disable the CGGA, consider the following points:

- Writing a 1 to any CGGA control word bits that are reserved can cause the system to crash. The Write command is described later in this appendix.
- Invalid data in a CGGA command can cause the system to crash.
- Executing a CGGA command changes the state of the DHiRes switch, altering the state of graphics screens. You must return the DHiRes switch to its original state when you are finished executing any CGGA command.
- If you speed up port 2, the Wait routine in firmware (see Appendix F) runs at 4 MHz rather than 1 MHz. Use the Wait routine in the code sample at the end of this appendix instead.
- If you speed up port 2, modem code might fail to work correctly.
- If you speed up ports 5 and 6, the disk drives no longer function.
- If the ROM isn't switched in before you execute a CGGA command, the system will crash. If RAM was switched in before you started, remember to return the RdLCRAM soft switch to its prior state before quitting.
- If you attempt to speed up the game paddles, they no longer function.
- Executing the Write command to the CGGA when the CGGA is disabled causes unpredictable results. You must be sure the CGGA is enabled before executing the Write command.
- There is no way to determine the state of the system's speed at any given time—many factors cause it to change frequently.
- If you execute a command to the CGGA on any machine other than the Apple IIc Plus, the system will crash.
- Making changes to the state of the CGGA can cause other applications to work incorrectly.
- Any or all of the above caveats may change with future revisions of the Apple IIc Plus hardware and firmware.

If your program makes any changes in CGGA settings, you *must* restore the CGGA to its original state before your program exits. Any changes in CGGA settings can prevent another application from running properly. Each time the system is reset, the Reset handler returns the CGGA to the default settings described at the beginning of this section; if your application quits by resetting the system, your program does not have to reset the CGGA.

CGGA commands

This section describes how to make calls to the CGGA to enable it, disable it, and change its mode of operation.

▲ **Warning** Modifying the accelerator registers without a full understanding of the CGGA and the Apple IIc Plus hardware and firmware can render the system inoperative, requiring the user to shut the machine off and turn it back on again to regain control. ▲

To send a command to the CGGA, you must first push the command parameters onto the stack, then execute a JSR instruction to the accelerator entry point, $C7C7. The parameters consist of a pointer to a buffer (when necessary) and a command number. For your convenience, sample code is provided with each command description showing the proper way to set up the parameters for that call.

The CGGA firmware pulls the parameters off the stack and checks the command number to determine if it corresponds to a valid command. If the command number is valid, the firmware performs the function specified by the command and returns to the calling routine with a value of $00 in the A register (accumulator). If the command number is not valid, the firmware returns the value $01 in the A register to indicate an error. In either case, the c (carry) flag is set. The calls themselves do not return errors. If the command number is valid, the firmware assumes that the parameters provided are also valid.

Before sending a command to the CGGA, you must be sure that the lower half of the ROM (the main ROM) is selected and that the ROM is switched in. To determine whether the main ROM is selected, check the contents of location $FCFF. If $FCFF contains a nonzero value, the main ROM is selected.

▲ **Warning** The system will crash if you send a command to the CGGA when the main ROM is not both selected and switched in. ▲

$01 Enable Accelerator

Description It is possible for an application program to disable the CGGA completely. The Enable Accelerator command reenables the CGGA.

Command number $01

Parameter list Command number

Example

```
lda     #$01                ;Enable Accelerator command
pha                         ;Command pushed on stack
jsr     Accelerator_Entry   ;Jump to the Accelerator
                            ;entry point
```

$02 Disable Accelerator

Description The Disable Accelerator command disables the CGGA completely. The Apple IIc Plus operates at 1 MHz as if there were no CGGA chip installed.

Command number $02

Parameter list Command number

Example

```
lda     #$02                ;Disable Accelerator command
pha                         ;Command pushed on stack
jsr     Accelerator_Entry   ;Jump to the Accelerator
                            ;entry point
```

$03 Lock Accelerator

Description The Lock Accelerator command locks the CGGA so that it cannot receive any commands except for the Unlock Accelerator command.

Command number $03

Parameter list Command number

Example

```
lda     #$03                ;Lock Accelerator command
pha                         ;Command pushed on stack
jsr     Accelerator_Entry   ;Jump to the Accelerator
                            ;entry point
```

$04 Unlock Accelerator

Description The Unlock Accelerator command reverses the effect of the Lock Accelerator command, making it possible for the CGGA to accept all commands.

Command number $04

Parameter list Command number

Example

```
lda     #$04                ;Unlock Accelerator command
pha                         ;Command pushed on stack
jsr     Accelerator_Entry   ;Jump to the Accelerator
                            ;entry point
```

$05 Read Accelerator

Description

The Read Accelerator command reads the CGGA registers, codes the state of the registers into a 2-byte word known as the *control word,* and places the control word in a buffer defined by the calling routine. The coding for the control word is shown in Table H-1.

■ **Table H-1** Accelerator control word

Bit	Meaning
Low byte	
7	Speaker speed (1 = fast)
6	Port 7 speed (1 = fast)
5	Port 6 speed (1 = fast)
4	Port 5 speed (1 = fast)
3	Port 4 speed (1 = fast)
2	Port 3 speed (1 = fast)
1	Port 2 speed (1 = fast)
0	Port 1 speed (1 = fast)
High byte	
7	Reserved
6	Paddle speed (1 = slow)
5	Reserved
4	Reserved
3	CGGA enable (1 = disabled)*
2	Reserved
1	Reserved
0	Reserved

* This bit is set and cleared by the Disable Accelerator and Enable Accelerator commands, not by the Write Accelerator command. For the Write Accelerator command, this bit is reserved.

Command number $05

Parameter list Command number

Pointer to buffer in which to store the control word

Example

```
lda     #<buffer              ;High byte of buffer address
pha                           ;Byte pushed on stack
lda     #>buffer              ;Low byte of buffer address
pha                           ;Byte pushed on stack
lda     #$05                  ;Read Accelerator command
pha                           ;Command pushed on stack
jsr     Accelerator_Entry     ;Jump to the Accelerator
                              ;entry point
```

$06 Write Accelerator

Description

The Write Accelerator command sends a control word to the CGGA, resetting the values in the CGGA's internal registers. You can use this command to control which ports in the Apple IIc run at 1 MHz and which run at 4 MHz. The CGGA control word is shown in Table H-1. Notice that you use the Enable Accelerator and Disable Accelerator commands to set or clear bit 3 of the high byte; do not write to this bit. All other bits are set by the Write Accelerator command.

△ **Important** Executing the Write command to the CGGA when the CGGA is disabled causes unpredictable results. Use the Read command—and then the Enable command if necessary—to make sure the CGGA is enabled before executing the Write command. △

Command number $06

Parameter list

Command number
Pointer to buffer in which the control word is stored

Example

```
lda     #<buffer            ;High byte of buffer address
pha                         ;Byte pushed on stack
lda     #>buffer            ;Low byte of buffer address
pha                         ;Byte pushed on stack
lda     #$06                ;Write Accelerator command
pha                         ;Command pushed on stack
jsr     Accelerator_Entry   ;Jump to the Accelerator
                            ;entry point
```

Code sample

The following code sample constitutes a shell that goes around any calls you make to the CGGA. Several routines are provided; use only those that you need. The code sample includes the following routines:

- The main routine that calls the other subroutines.
- A routine that checks the ID bytes of the computer to determine if it is an Apple IIc Plus. If you are certain that the machine is an Apple IIc Plus, you don't have to check it again, but remember that any CGGA call causes any Apple II computer *other* than an Apple IIc Plus to crash.
- A routine that saves the states of the DHiRes and 80Col soft switches and turns off 80-column mode before you send any commands to the CGGA. If you can make your calls to the CGGA at the beginning of your program before setting the DHiRes switch and 80Col switch, then you do not have to use this routine. Just be sure to set the DHiRes and 80Col switches to the settings you want after you have finished sending commands to the CGGA. Remember also to switch in the main ROM before sending any commands to the CGGA, and to return the RdLCRAM soft switch to its prior state when you are finished.
- A routine that restores the saved state of the machine.
- A routine that unlocks the CGGA so that it can receive commands. You must use this routine before sending any commands to the CGGA.
- A routine that locks the CGGA so that no additional commands can be sent to it during normal system use. You must use this routine before quitting.
- A routine that you can use instead of the firmware version of the Wait routine (described in Appendix F) if your program speeds up port 2.

```
*******************************************************
*using the accelerator:
*
*This code implements any of the calls documented
*in this section that talk to the accelerator in the IIc Plus.
*
*               Entry:   ROM must be enabled
*
*               Exit:    C=0
*                        A=0
*                        X,Y undefined
********************************************************
use.accel
accel.entry     equ      $C7C7              ;entry point to talk to accelerator

                jsr      check.id           ;must be at least a IIc Plus
                bcc      dont               ;won't work on any other machine
                jsr      save.state         ;save state if you don't know what it is

                jsr      unlock             ;must unlock it before anything else
*               .
*               .
*add your call(s) here.  See descriptions of calls for format
*               .
*               .
                jsr      lock               ;must lock it when all finished

                jsr      restore.state      ;must restore it if you saved it
dont            rts                         ;go back to the calling routine or user
```

```
***********************************************************
*check.id:      Calls to the accelerator can ONLY be made on
*               a IIc Plus.  If this call is made on any other
*               Apple II machine, the program WILL crash!
*               This routine checks for the current IIc Plus
*               and beyond (for future compatability).
*
*               Entry:   ROM must be enabled
*
*               Exit:    C=0 if not IIc Plus
*                        C=1 if IIc Plus
*                        A,X,Y undefined
***********************************************************
check.id
id1             equ      $FBB3
id2             equ      $FBC0
id3             equ      $FBBF

                lda      id1                ;should be $06
                cmp      #$06
                bne      no                 ;if not IIc Plus then quit

                lda      id2
                bne      no                 ;should be $00 -- if not then quit

                lda      id3                ;should be $≥ 5
                bmi      no                 ;make sure it's not the orig IIc with #$FF
                cmp      #$05
                bcs      done               ;if c=0 then it failed-- make sure it does!
no
                clc                         ;clear carry means wrong machine
done            rts                         ;go back with status of test
```

```
****************************************************************
*save/unsave state:      This routine saves and restores the states
*                        of the double hires and 80 column soft
*                        switches.  It turns off 80 column mode
*                        so that talking to the accelerator can't
*                        accidently turn on double hires.
*
*NOTE:                   If you put any calls to the accelerator
*                        BEFORE you set the state of the machine
*                        at the beginning of your program, then
*                        this routine will not be necessary.  Simply
*                        make sure your initialization routine
*                        sets the 80 column switch and the double
*                        hires switches to the state your program
*                        requires after making any calls to the
*                        accelerator.
*
*               Entry:   nothing
*
*               Exit:    A scrambled
*                        80Col firmware is turned off
*                        X,Y unchanged
****************************************************************
save.state

rd80col         equ      $C01F               ;if bit 7 =1 then 80col is on
rddhires        equ      $C07F               ;if bit 7 = 0 then dhires is on
on.80col        equ      $C00D               ;writing turns on 80 col
off.80col       equ      $C00C               ;writing turns off 80 col
on.dhires       equ      $C05E               ;writing turns on dhires
off.dhires      equ      $C05F               ;writing turns off dhires

                lda      rd80col             ;see if 80 column is on
                asl      a                   ;save state
                bcc      @1                  ;if branch then it's off
                sta      off.80col           ;turn it off first
@1
                lda      rddhires            ;see if double hires on
                php                          ;save c and n flags on stack
                pla
                sta      temp                ;store result of both tests
                rts                          ;return to caller

unsave.state
                lda      temp                ;get back the status flags
                pha
                plp                          ;want status flags back in P reg
                bcc      @1                  ;branch if 80Col should be off
                                             ;because it already is
                sta      on.80col            ;should be on - so turn it on
@1              bmi      @2                  ;branch if dhires is off
                sta      on.dhires           ;restore dhires to on
                bpl      done2               ;branch unconditionally
@2              sta      off.dhires          ;restore dhires to off
done2           rts

temp            dfb      $00                 ;used for temp storing of states
```

```
*************************************************************************
*unlock:         Unlocks the accelerator so it can be accessed.  This
*                must be done before anything else.
*
*NOTE:           Please note that the state of some soft switches
*                will be affected by this call.  (See save/restore.state)
*
*Entry:          nothing
*
*Exit:           C=0
*                A=0
*************************************************************************
unlock
                 lda      #$04              ;cmd to unlock
                 pha                        ;store it on stack
                 jsr      accel.entry       ;call ROM
                 rts                        ;back to caller
*************************************************************************
*lock:           Locks the accelerator so that normal system
*                use cannot affect it.  This must be the last
*                call made to the accelerator.
*
*Entry:          nothing
*
*Exit:           C=0
*                A=0
*************************************************************************
lock
                 lda      #$03              ;cmd to lock accelerator
                 pha                        ;store it on stack
                 jsr      accel.entry       ;call ROM
                 rts                        ;back to caller
```

```
*********************************************************************
*Wait.ram:        This replaces the ROM version of the wait routine
*                 if your program speeds up port 2.
*
*NOTE:            Your program must call this routine and not the one in
*                 ROM.  You may put this in the language card and disable
*                 the ROM if you are not using any other ROM routines.
*
*NOTE:            This routine will run correctly at either fast or normal
*                 speed.  It can also be run on any other version of the IIc
*                 or IIe without harm.
*
*Entry:           A=$00 - $FF depending on the amount of time to wait:
*                 Min delay = 1/2(50+25A+5A^2)+29
*                 Delay is at least as great as caused by the wait routine in ROM,
*                 and in most cases is exactly the same.
*                 This routine has A-12 fewer cycles than the wait routine
*                 in ROM.
*                 X,Y,P undefined
*
*Exit:            X,Y unchanged
*                 A=$00
*                 P undefined
*********************************************************************
wait.ram

kbd               equ      $C000              ;address offset for port I/O

                  phy                         ;save X and Y
                  ldy      #$D0               ;$C0D0 is guaranteed 50 ms slow
                  phx                         ;on IIc Plus but won't hurt other
                  sec                         ;IIc or IIe's
                  txa                         ;save wait value
@1
                  lda      kbd,Y              ;this starts the slow down
                  txa                         ;new version of wait routine
@2
                  sbc      #$01               ;min delay = 1/2(50+25A+5A^2)+29
                  bne      @2                 ;timing is at least that of the old
                  dex                         ;one and in most cases exact timing!
                  bne      @1
                  plx                         ;restore X and Y
                  ply
                  rts
```

Glossary

accumulator: See **A register.**

ACIA: See **Asynchronous Communications Interface Adapter.**

acronym: A word formed from the initial letters of a name or phrase, such as ROM (from **read-only memory**).

ADC: See **analog-to-digital converter.**

address: A number that specifies the location of a single byte of memory. Addresses can be given as decimal integers or as hexadecimal integers. A 64 KB system has addresses ranging from 0 to 65535 (in decimal) or from $0000 to $FFFF (in hexadecimal).

algorithm: A step-by-step procedure for solving a problem or accomplishing a task.

alternate character set: The set of characters displayed on the screen of an Apple IIc family computer when the enhanced video firmware is active. Compare with **primary character set.**

American Simplified Keyboard: See **Dvorak keyboard.**

analog: Varying smoothly and continuously over a range, rather than changing in discrete jumps. For example, a conventional 12-hour clock face is an analog device that shows the time of day by the continuously changing position of the clock's hands. Compare **digital.**

analog data: Data in the form of continuously variable quantities. Compare **digital data.**

analog signal: A signal that varies continuously over time, rather than being sent and received in discrete intervals. Compare **digital signal.**

analog-to-digital converter (ADC): A device that converts quantities from analog to digital form. For example, computer hand controls convert the position of the control dial (an analog quantity) into a discrete number (a digital quantity) that changes stepwise even when the dial is turned smoothly.

AND: A Boolean (logical) operator that produces a true result (1) if both its operands are true, and a false result (0) if either or both its operands are false. Compare **OR, NOT, XOR.**

ANSI: Acronym for *American National Standards Institute,* which sets standards for many technical fields and is the most common standard for computer terminals.

Apple I: The first Apple computer. It was built in a garage in California by Steve Jobs and Steve Wozniak.

Applesoft BASIC: The Apple II dialect of the BASIC programming language. An interpreter for creating and executing Applesoft BASIC programs is built into the firmware of computers in the Apple II family. See also **BASIC, Integer BASIC.**

Apple II: A family of personal computers, including the original Apple II, the Apple II Plus, the Apple IIe, the Apple IIc, the Apple IIc Plus, and the Apple IIGS. The original Apple II used Integer BASIC instead of Applesoft BASIC, and it required a keyboard command (PR#6) in order to start up from a disk.

Apple IIc: A transportable personal computer in the Apple II family with a disk drive and 80-column display capability built in. There are three versions of the Apple IIc computer: the original Apple IIc, the UniDisk 3.5 Apple IIc, and the memory expansion Apple IIc. See also **Apple IIc Family, Apple IIc Plus.**

Apple IIc family: A family of personal computers, including the original Apple IIc, the UniDisk 3.5 Apple IIc, the memory expansion Apple IIc, and the Apple IIc Plus. See also **Apple IIc, Apple IIc Plus.**

Apple IIc Plus: The latest member of the Apple IIc family, with a 3.5-inch disk drive, 80-column display, and memory expansion capability built in. The Apple IIc Plus is capable of running at up to 4 MHz processor speed, as compared to the 1 MHz speed of other members of the Apple IIc family. See also **Apple IIc , Apple IIc Family.**

Apple IIe: A personal computer in the Apple II family with seven expansion slots and an auxiliary memory slot that allow the user to enhance the computer's capabilities with peripheral and auxiliary cards. The Apple IIe has been improved and enhanced over the years.

Apple IIe 80-Column Text Card: A peripheral card that plugs into the auxiliary memory slot of the Apple IIe and allows the computer to display either 40 or 80 characters per line.

Apple IIe Extended 80-Column Text Card: A peripheral card that plugs into the auxiliary memory slot of the Apple IIe and allows the computer to display either 40 or 80 characters per line while extending the computer's memory capacity by 64 KB.

Apple IIGS: A personal computer in the Apple II family. The Apple IIGS uses a 16-bit microprocessor and has 256 KB of RAM. It has slots like the Apple IIe and ports like the Apple IIc, and contains a 15-voice custom sound chip.

Apple II Pascal: A software system for the Apple II family that lets you create and execute programs written in the Pascal programming language. Apple II Pascal was adapted by Apple Computer from the University of California, San Diego, Pascal Operating System (UCSD Pascal).

Apple II Plus: A personal computer in the Apple II family with expansion slots that allow the user to enhance the computer's capabilities with peripheral and auxiliary cards.

Apple III: An Apple computer that is part of the Apple II family. The Apple III offered a built-in disk drive and built-in RS-232-C (serial) port. Its memory was expandable to 256 KB.

application program: A program written for some specific purpose, such as word processing, database management, graphics, or telecommunication. Compare **system program.**

A register: An 8-bit register in the 65C02 microprocessor that the microprocessor uses to perform logical and arithmetic calculations and to store results of such calculations. Also called the *accumulator*.

argument: A value on which a function or statement operates; it can be a number or a variable. For example, in the BASIC statement VTAB 10, the number 10 is the argument. Compare **operand.**

arithmetic expression: A combination of numbers and arithmetic operators (such as 3 + 5) that indicates some operation to be carried out.

arithmetic operator: An operator, such as +, that combines numeric values to produce a numeric result. Compare **logical operator, relational operator.**

ASCII: Acronym for *American Standard Code for Information Interchange;* pronounced "ASK-ee." A code in which the numbers from 0 to 127 stand for text characters. ASCII code is used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device. Compare **EBCDIC.**

ASIC: Acronym for *application-specific integrated circuit.* A custom integrated circuit.

assembler: A language translator that converts a program written in assembly language into an equivalent program in machine language. The opposite of a **disassembler.**

assembly language: A low-level programming language in which individual machine-language instructions are written in a symbolic form that's easier to understand than machine language itself. Each assembly-language instruction produces one machine-language instruction. See also **machine language.**

asynchronous: Not synchronized by a mutual timing signal or clock. Compare **synchronous.**

Asynchronous Communications Interface Adapter (ACIA): An integrated circuit that provides the control signals and parallel/serial conversion for a port.

asynchronous transmission: A method of data transmission in which the receiving and sending devices don't share a common timer, and no timing data is transmitted. Each information character is individually synchronized, usually by the use of start and stop bits. The time interval between characters isn't necessarily fixed. Compare **synchronous transmission.**

auxiliary slot: The special expansion slot inside the Apple IIe used for the Apple IIe 80-Column Text Card or Extended 80-Column Text Card, and also for the RGB monitor card. The slot is labeled AUX. CONNECTOR on the circuit board.

back panel: The rear surface of the computer, which includes the power switch, the power connector, and connectors for peripheral devices.

bandwidth: The range of frequencies a device can handle. Bandwidth and maximum data transfer rate are directly proportional. For example, a video monitor's greater bandwidth allows it to display more information per scan frame than most home television sets can. To display 80 columns of text, a monitor should have a bandwidth of at least 12 MHz.

bank: A portion of memory. In this book, the term *bank* is used only to refer to the address space from $D000 to $DFFF, which can be switched between four portions of memory (*main RAM Bank 1, main RAM Bank 2, auxiliary RAM Bank 1,* and *auxiliary RAM Bank 2*).

bank-switched memory: All memory that uses the address space from $D000 to $DFFF. See also **bank.**

base address: In *indexed addressing,* the fixed component of an address.

BASIC: Acronym for *Beginners All-purpose Symbolic Instruction Code.* BASIC is a high-level programming language designed to be easy to learn. Two versions of BASIC are available from Apple Computer for use with all Apple II–family systems: Applesoft BASIC (built into the firmware) and Integer BASIC.

baud: A unit of data transmission speed: the number of discrete signal state changes per second. Often, but not always, equivalent to *bits per second.* Compare **bit rate.**

binary: Characterized by having two different components, or by having only two alternatives or values available; sometimes used synonymously with *binary system.*

binary digit: The smallest unit of information in the binary number system; a 0 or a 1. Also called a *bit.*

binary operator: An operator that combines two operands to produce a result. For example, + is a binary arithmetic operator; < is a binary relational operator; OR is a binary logical operator. Compare **unary operator.**

binary system: The representation of numbers in the base-2 system, using only the two digits 0 and 1. For example, the numbers 0, 1, 2, 3, and 4 become 0, 1, 10, 11, and 100 in binary notation. The binary system is commonly used in computers because the values 0 and 1 can easily be represented in a variety of ways, such as the presence or absence of current, positive or negative voltage, or a white or black dot on the display screen. A single binary digit—a 0 or a 1—is called a *bit.* Compare **decimal, hexadecimal.**

bit: A contraction of *binary digit.* The smallest unit of information that a computer can hold. The value of a bit (1 or 0) represents a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing. See also **binary system.**

bit rate: The speed at which bits are transmitted, usually expressed as *bits per second,* or *bps.* Compare **baud.**

bits per second: See **bit rate.**

block: A number of bytes of data treated as a unit. A block may be made up of any number of bytes, but in the Apple II family, a standard block is 512 bytes. See also **block device.**

block device: A peripheral device that executes I/O operations by grouping data into bundles called *blocks.*

board: See **printed-circuit board.**

body: In BASIC, the statements or instructions that make up a part of a program, such as a loop or a subroutine.

boot: Another way to say *start up.* A computer boots by loading a program into memory from an external storage medium such as a disk. Starting up is often accomplished by first loading a small program, which then reads a larger program into memory. The program is said to "pull itself up by its own bootstraps"—hence the term *bootstrapping* or *booting.*

boot disk: See **startup disk.**

bootstrap: See **boot.**

bps: See **bit rate.**

branch: (v) To pass program control to a line or statement other than the next in sequence. (n) A statement that performs a branch. See also **conditional branch, unconditional branch.**

BREAK: A SPACE (0) signal, sent over a communication line, of long enough duration to interrupt the sender. This signal is often used to end a session with a time-sharing service. BREAK is also used in BASIC to stop execution of a program; it's generated by pressing Control-C.

BRK: A "software interrupt." An instruction that causes the 6502 or 65C02 microprocessor to halt. Pronounced "break."

buffer: (1) An area in memory that is allocated to store data on a temporary basis. (2) A dedicated memory where information can be stored by one program or device and then read at a different rate by another.

bug: An error in a program that causes it not to work as intended. The expression reportedly comes from the early days of computing when an itinerant moth shorted a connection and caused a breakdown in a room-size computer.

bus: A group of wires or circuits that transmit related information from one part of a computer system to another. In a network, a line of cable with connectors linking devices together. A bus network has a beginning and an end. (It's not in a closed circle or T shape.)

byte: A unit of information consisting of a fixed number of bits. On Apple II systems, 1 byte consists of a series of 8 bits, and a byte can represent any value between 0 and 255. The sequence represents an instruction, letter, number, punctuation mark, or other character. See also **kilobyte, megabyte.**

cable: An insulated bundle of wires with connectors on the ends; the number of wires varies with the type of connection. Examples are serial cables, disk drive cables, and LocalTalk cables.

cache glue gate array (CGGA): A custom integrated circuit in the Apple IIc Plus computer; it implements a RAM cache for the CPU and provides the clock for the CPU.

call: (v) To request the execution of a subroutine, function, or procedure. (n) A request from the keyboard or from a procedure to execute a named procedure.

carriage return: An ASCII character (decimal 13) that ordinarily causes a printer or display device to place the next character on the left margin.

carrier: The background signal on a communication channel that is modified to carry information. Under RS-232-C rules, the carrier signal is equivalent to a continuous MARK (1) signal; a transition to 0 then represents a start bit.

carry flag: A status bit in the 6502 or 65C02 microprocessor, used as a ninth bit with the eight A register bits in addition, subtraction, rotation, and shift operations.

cathode-ray tube (CRT): An electronic device, such as a television picture tube, that produces images on a phosphor-coated screen. The phosphor coating emits light when struck by a focused beam of electrons. A common display device used with personal computers.

central processing unit (CPU): The "brain" of the computer; the microprocessor that performs the actual computations in machine language. See also **microprocessor.**

character: Any symbol that has a widely understood meaning and thus can convey information. Some characters—such as letters, numbers, and punctuation—can be displayed on the monitor screen and printed on a printer. Compare **control character.**

CGGA: See **cache glue gate array.**

character code: A number used to represent a character for processing by a computer system.

character set: The entire set of characters that can be either shown on a monitor or used to code computer instructions. In a printer, the entire set of characters that the printer is capable of printing.

chip: See **integrated circuit.**

Clear To Send: An RS-232-C signal from a DCE to a DTE that is normally kept false until the DCE makes it true, indicating that all circuits are ready to transfer data out. See also **Data Communication Equipment, Data Terminal Equipment.**

CMOS: Acronym for *complementary metal-oxide semiconductor.* A type of integrated circuit that requires less power to operate than ICs based on other technologies.

code: (1) A number or symbol used to represent some piece of information. (2) The statements or instructions that make up a program.

cold start: The process of starting up the Apple II when the power is first turned on (or as if the power had just been turned on) by loading the operating system into main memory, and then loading and running a program. Compare **warm start.**

column: A vertical arrangement of graphics points or character positions on the display.

command: An instruction that causes the computer to perform some action. A command can be typed from a keyboard, selected from a menu with a hand-held device (such as a mouse), or embedded in a program.

command character: An ASCII character, usually Control-A or Control-I, that causes the serial port firmware to interpret subsequent characters as commands.

Command key: A **control** key on Apple II–family keyboards marked with the outline of an apple. This key was formerly called the *Open Apple key.*

command register: An ACIA location (at $C09A for port 1 and $C0AA for port 2) that stores parity type and RS-232-C signal characteristics.

compiler: A language translator that converts a program written in a high-level programming language (source code) into an equivalent program in some lower-level language such as machine language (object code) for later execution. Compare **interpreter.**

composite video: A video signal that includes both display information and the synchronization (and other) signals needed to display it.

computer: An electronic device that performs predefined (programmed) computations at high speed and with great accuracy. A machine that is used to store, transfer, and transform information.

computer language: See **programming language.**

computer system: A computer and its associated hardware, firmware, and software.

conditional branch: A branch whose execution depends on the truth of a condition or the value of an expression. Compare **unconditional branch.**

configuration: (1) The total combination of hardware components—CPU, video display device, keyboard, and peripheral devices—that make up a computer system. (2) The software settings that allow various hardware components of a computer system to communicate with each other.

connector: A plug, socket, jack, or port.

constant: In a program, a symbol that represents a fixed, unchanging value. Compare **variable.**

control character: A nonprinting character that controls or modifies the way information is printed or displayed. In the Apple II family, control characters have ASCII values between 0 and 31 ($00 to $1F) and are typed from a keyboard by holding down the Control key while pressing some other key.

control code: One or more nonprinting characters—included in a text file—whose function is to change the way a printer prints the text. For example, a program may use certain control codes to turn boldface printing on and off. Compare **control character.**

control key: A general term for a key that controls the operation of other keys; for example, Command, Caps Lock, Control, Option, and Shift. When you hold down or engage a control key while pressing another key, the combination makes that other key behave differently. Also called a *modifier key.*

Control key: A specific key on Apple II–family keyboards that produces control characters when used in combination with other keys.

controller card: A peripheral card that connects a device such as a printer or disk drive to a computer's main logic board and controls the operation of the device.

control register: An ACIA location (at $C09B for port 1 and $C0AB for port 2) that stores data format and baud rate selections.

Control-Reset: A combination keystroke on Apple II–family computers that usually causes an Applesoft BASIC program or command to stop immediately. If a program disables the Control-Reset feature, you must turn the computer off to get the program to stop.

copy protect: To make a disk uncopyable. Software publishers frequently try to copy protect their disks to prevent them from being illegally duplicated by software pirates. Compare **write protect.**

CPU: See **central processing unit.**

CRT: See **cathode-ray tube.**

CTS: See **Clear To Send.**

crash: To cease to operate unexpectedly, possibly destroying information in the process.

current input device: The source, such as the keyboard or a modem, from which a program is currently receiving its input.

current output device: The destination, such as the display screen or a printer, currently receiving a program's output.

cursor: A symbol displayed on the screen marking where the user's next action will take effect or where the next character typed from the keyboard will appear.

DAC: See **digital-to-analog converter.**

data: Information, especially information used or operated on by a program. The smallest unit of information a computer can understand is a bit.

data bits: The bits in a communication transfer that contain information. Compare **start bit, stop bit.**

Data Carrier Detect (DCD): An RS-232-C signal from a DCE (such as a modem) to a DTE (such as an Apple II) indicating that a communication connection has been established.

Data Communication Equipment (DCE): As defined by the RS-232-C standard, any device that transmits or receives information. Usually this device is a modem.

data format: The form in which data is stored manipulated, or transferred.

Data Set Ready (DSR): An RS-232-C signal from a DCE to a DTE indicating that the DCE has established a connection.

Data Terminal Equipment (DTE): As defined by the RS-232-C standard, any device that generates or absorbs information, thus acting as an endpoint of a communication connection. A computer might serve as a DTE.

Data Terminal Ready (DTR): An RS-232-C signal from a DTE to a DCE indicating a readiness to transmit or receive data.

DCD: See **Data Carrier Detect.**

DCE: See **Data Communication Equipment.**

debug: A colloquial term that means to locate and correct an error or the cause of a problem or malfunction in a computer program. Compare **troubleshoot.** See also **bug.**

decimal: The common form of number representation used in everyday life, in which numbers are expressed in in the base-10 system, using the ten digits 0 through 9. Compare **binary, hexadecimal.**

default: A preset response to a question or prompt. The default is automatically used by the computer if you don't supply a different response. Default values prevent a program from stalling or crashing if no value is supplied by the user.

Delete key: A key on the upper-right corner of the Apple IIe, IIc, IIc Plus, and IIGS keyboards that erases the character immediately preceding (to the left of) the cursor. Similar to the Macintosh Backspace key.

demodulate: To recover the information being transmitted by a modulated signal. For example, a conventional radio receiver demodulates an incoming broadcast signal to convert it into the sound emitted by the radio's speaker. Compare **modulate.**

device: (1) A component on the logic board that can exchange information with the CPU. The IWM chip is an example of a device. (2) A peripheral device.

device driver: A program that manages the transfer of information between the computer and a peripheral device.

digit: (1) One of the characters 0 through 9, used to express numbers in decimal form. (2) One of the characters used to express numbers in some other form, such as 0 and 1 in binary, or 0 through 9 and A through F in hexadecimal.

digital: Represented in a discrete (noncontinuous) form, such as numerical digits or integers. For example, contemporary digital clocks show the time as a digital display (such as 2:57) instead of using the positions of a pair of hands on a clock face. Compare **analog.**

digital data: Data that can be represented by digits—that is, data that are discrete rather than continuously variable. Compare **analog data.**

digital signal: A signal that is sent and received in discrete intervals. A signal that does not vary continuously over time. Compare **analog signal.**

digital-to-analog converter: A device that converts quantities from digital to analog form.

DIP: See **dual in-line package.**

DIP switches: A bank of tiny switches, each of which can be moved manually one way or the other to represent one of two values (usually on and off). See also **dual in-line package.**

disassembler: A language translator that converts a machine-language program into an equivalent program in assembly language, which is easier for programmers to understand. The opposite of an **assembler.**

disk: An information-storage medium consisting of a flat, circular, magnetic surface on which information can be recorded in the form of small magnetized spots, in a manner similar to the way sounds are recorded on tape. See **floppy disk, hard disk.**

disk-based: See **disk-resident.**

disk controller card: A peripheral card that provides the connection between one or two disk drives and the computer. This connection, or interface, is built into both the Apple IIc–family and Macintosh-family computers.

disk controller unit: See **IWM.**

disk drive: The device that holds a disk, retrieves information from it, and saves information to it.

disk envelope: A removable, protective paper sleeve used when handling or storing a 5.25-inch disk. It must be removed before you insert the disk in a disk drive. Compare **disk jacket.**

disk jacket: A permanent, protective covering for a disk. 5.25-inch disks have flexible paper or plastic jackets; 3.5-inch disks have hard plastic jackets. The disk is never removed from the jacket. Compare **disk envelope.**

disk-resident: An adjective describing a program that does not remain in memory. The computer retrieves all or part of the program from the disk, as needed. Sometimes called *disk-based.* Compare **memory-resident.**

Disk II drive: An older type of disk drive made and sold by Apple Computer for use with the Apple II, II Plus, and IIe. It uses 5.25-inch disks.

display: (1) A general term to describe what you see on the screen of your display device when you're using a computer; from the verb form, which means "to place into view." (2) Short for a display device.

display color: The color currently being used to draw graphics on the display screen.

display device: A device that displays information, such as a television set or video monitor.

display screen: The screen of the monitor; the area where you view text and pictures when using the computer.

DOS 3.2: An early Apple II operating system. DOS stands for *disk operating system;* 3.2 is the version number. Disks formatted using DOS 3.2 have 13 sectors per track.

DOS 3.3: An operating system used by the Apple II family of computers. DOS stands for *disk operating system;* 3.3 is the version number. Disks formatted with DOS 3.3 have 16 sectors per track.

Double Hi-Res graphics: A high-resolution display mode on the Apple II family of computers, consisting of an array of points 560 wide by 192 high, with 16 colors. When a text window is in use, the visible Hi-Res graphics display is 560 by 160 points. Compare **Lo-Res graphics, Hi-Res graphics.**

drive: See **disk drive.**

DSR: See **Data Set Ready.**

DTE: See **Data Terminal Equipment.**

DTR: See **Data Terminal Ready.**

dual in-line package (DIP): An integrated circuit packaged in a narrow rectangular box with a row of metal pins along each side. Switches on the box allow you to change settings. For example, ImageWriter® printer DIP switches control functions such as line feed, form length, and baud setting.

Dvorak keyboard: An alternate keyboard layout, also known as the *American Simplified Keyboard,* which increases typing speed because the keys most often used are in the positions easiest to reach. Compare **Sholes keyboard.**

EBCDIC: Acronym for *Extended Binary-Coded Decimal Interchange Code;* pronounced "EB-si-dik." A code used by IBM that represents each letter, number, special character, and control character as an 8-bit binary number. EBCDIC has a character set of 256 8-bit characters. Compare **ASCII.**

effective address: In machine-language programming, the address of the memory location on which a particular instruction operates, which may be arrived at by indexed addressing or some other addressing method.

80-column text card: A peripheral card that allows the Apple II, Apple II Plus, and Apple IIe to display text in either 40 columns or 80 columns.

80/40-column switch: A switch that controls the maximum number of columns or characters across the screen. A television can legibly display a maximum of 40 characters across the screen, whereas a video monitor can display 80 characters.

end-of-line character: A character which indicates that the preceding text constitutes a full line.

enhanced video firmware: The firmware in the Apple IIc computers that performs the same functions as the firmware on an 80-column card in an Apple IIe. The enhanced video firmware implements the alternate character set, the 80-column screen, and a variety of I/O routines not available in the standard video firmware.

error code: A number or other symbol representing a type of error.

error message: A message displayed or printed to tell you of an error or problem in the execution of a program or in your communication with the system. An error message is often accompanied by a beep.

escape character: ASCII character $1B (Esc). Pressing either the Esc key or Control-[generates an escape character.

escape code: A sequence of characters that begins with an escape character ($1B) and constitutes a complete command. Usually synonymous with *escape sequence.*

Escape key: See **Esc key.**

escape mode: A state of the Apple IIe and IIc entered by pressing the Esc key when certain programs are running. Subsequent keystrokes take on special meanings for positioning the cursor and controlling the display of text on the screen.

escape sequence: A sequence of keystrokes, beginning with the Esc key. In escape mode, escape sequences are used for positioning the cursor and controlling the display of text on the screen. Escape sequences are also used as codes to control printers.

Esc key: A key on Apple II–family computers that generates the escape character ($1B). In many applications, pressing Esc allows you to return to a previous menu or to stop a procedure.

even/odd parity check: In data transmission, a check that tests whether the number of 1 bits in a group of binary digits is even (even parity check) or odd (odd parity check).

even parity: In data transmission, the use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits an even number; used as a means of error checking. Compare **MARK parity, odd parity, SPACE parity.**

exclusive OR: See **XOR.**

execute: To perform the actions specified by a program command or sequence of commands.

expansion slot: A connector into which you can install a peripheral card. Sometimes called a *peripheral slot*. See also **auxiliary slot.**

expression: A formula in a program that defines a calculation to be performed.

FIFO: Acronym for "first in, first out" order, which is a storage and retrieval method in which the first item stored is the first item retrieved.

file: Any named, ordered collection of information stored on a disk. Application programs and operating systems on disks are files. You make a file when you create text or graphics, give the material a name, and save it to disk.

firmware: Programs stored permanently in read-only memory (ROM). Such programs (for example, the Applesoft Interpreter and the System Monitor) are built into the computer at the factory. They can be executed at any time but cannot be modified or erased from main memory. Compare **hardware, software.**

flag: A variable whose value (usually 1 or 0, standing for *true* or *false*) indicates whether some condition holds or whether some event has occurred. A flag is used to control the program's actions at some later time.

form feed: An ASCII character (decimal 12) that causes a printer or other paper-handling device to advance to the top of the next page.

Fortran: Short for *Formula Translator*. A high-level programming language especially suitable for applications requiring extensive numerical calculations, such as in mathematics, engineering, and the sciences.

framing error: In serial data transfer, the absence of the expected stop bit(s) at the end of a received character.

frequency: In alternating current (AC) signals, the number of complete cycles transmitted per second. Frequency is usually expressed in hertz (cycles per second), kilohertz (kilocycles per second), or megahertz (megacycles per second). In acoustics, frequency of vibration determines musical pitch.

full duplex: A four-wire communication circuit or protocol that allows two-way data transmission between two points at the same time. Compare **half duplex.**

function: A preprogrammed calculation that can be carried out on request from any point in a program. A function takes in one or more arguments and returns a single value. It can therefore be embedded in an expression.

game I/O connector: A 16-pin connector inside the Apple II, II Plus, and IIe, originally designed for connecting hand controls to the computer, but also used for connecting some other peripheral devices. Compare **hand control connector.**

general logic unit (GLU): A custom integrated circuit that performs a vaiety of general logic functions in the Apple IIc.

GCR: See **group code recording.**

global storage: An area reserved to keep global variables.

GLU: See **general logic unit.**

graphics: (1) Information presented in the form of pictures or images. (2) The display of pictures or images on a computer's display screen. Compare **text.**

group code recording (GCR): A method of encoding data in which each 3 bytes are written as four 8-bit patterns constructed so that no pattern contains more than two consecutive 0's.

half duplex: A two-wire communication circuit or protocol designed for data transmission in either direction but not both directions simultaneously. Compare **full duplex.**

hand control connector: A 9-pin connector on the back panel of the Apple IIe and IIc computers, used for connecting hand controls to the computer. Compare **game I/O connector.**

hand control: Peripheral devices with rotating dials and push buttons. Hand controls are used to control game-playing programs, but they can also be used in other applications.

hang: To cease operation because either an expected condition is not satisfied or an infinite loop is occurring. A computer that's hanging is called a *hung system*. Compare **crash.**

hard disk: A disk made of metal and sealed into a drive or cartridge. A hard disk can store very large amounts of information compared to a **floppy disk.**

hard disk drive: A device that holds a hard disk, retrieves information from it, and saves information to it. Hard disks made for microprocessors are permanently sealed into the drives.

hardware: In computer terminology, the machinery that makes up a computer system. Compare **firmware, software.**

hertz: The unit of frequency of vibration or oscillation, defined as the number of *cycles per second*. Named for the physicist Heinrich Hertz and abbreviated *Hz*.

hexadecimal: The representation of numbers in the base-16 system, using the ten digits 0 through 9 and the six letters A through F. For example, the decimal numbers 0, 1, 2, 3, 4, ... 8, 9, 10, 11, ... 15, 16, 17 would be shown in hexadecimal notation as 00, 01, 02, 03, 04, ... 08, 09, 0A, 0B, ... 0F, 10, 11. Hexadecimal numbers are easier for people to read and understand than are binary numbers, and they can be converted easily and directly to binary form. Each hexadecimal digit corresponds to a sequence of four binary digits, or bits. Hexadecimal numbers are usually preceded by a dollar sign ($).

high ASCII characters: ASCII characters with decimal values of 128 to 255. Called *high ASCII* because their high bit (first binary digit) is set to 1 (for *on*) rather than 0 (for *off*).

high-level language: A programming language that is relatively easy for people to understand. A single statement in a high-level language typically corresponds to several instructions of machine language. High-level languages available from Apple Computer include BASIC, Pascal, Instant Pascal, Logo, and Fortran. Compare **low-level language.**

high-order byte: The more significant half of a memory address or other 2-byte quantity. In the 6502 microprocessor used in the Apple II family of computers, the low-order byte of an address is usually stored first, and the high-order byte second. In the 68000 microprocessors used in the Macintosh family, the high-order byte is stored first.

Hi-Res graphics: A high-resolution display mode on the Apple II family of computers, consisting of an array of points 560 wide by 192 high, with six colors. When a text window is in use, the visible Hi-Res graphics display is 280 by 160 points. Compare **Lo-Res graphics, Double Hi-Res graphics.**

hold time: In computer circuits, the amount of time a signal must remain valid after some related signal has been turned off. Compare **setup time.**

Hz: See **hertz.**

IC: See **integrated circuit.**

IN#: This command designates the source of subsequent input characters. It can be used to designate a device in a slot or a machine-language routine as the source of input.

index: (1) A number used to identify a member of a list or table by its sequential position. (2) A list or table whose entries are identified by sequential position. (3) In machine-language programming, the variable component of an indexed address, contained in an index register and added to the base address to form the effective address.

indexed addressing: A method used in machine-language programming to specify memory addresses.

index register: A register in a computer processor that holds an index for use in indexed addressing. The 6502 microprocessor used in the Apple II family of computers has two index registers, called the **X register** and the **Y register.** The 68000 microprocessor used in Macintosh-family computers has 16 registers that can be used as index registers.

index variable: A variable whose value changes on each pass through a loop. Often called *control variable* or *loop variable.*

infinite loop: A section of a program that will repeat the same sequence of actions indefinitely.

initialized disk: A disk that has been organized into tracks and sectors by the computer and is therefore ready to store information.

input: Information transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem.

input/output (I/O): The process by which information is transferred between the computer's memory and its keyboard or peripheral devices.

input/output unit (IOU): The custom integrated circuit that implements several soft switches, detects mouse movement, and generates timing and control signals for the video circuits, the speaker, and the keyboard in the Apple IIc.

input routine: A machine-language routine; the standard input routine reads characters from the keyboard. A different input routine might, for example, read them from an external terminal.

instruction: A unit of a machine-language or assembly-language program corresponding to a single action for the computer's processor to perform.

integer: A whole number in fixed-point form. Compare **real number.**

Integer BASIC: A version of the BASIC programming language used by the Apple II family of computers. Integer BASIC is older than Applesoft BASIC and is capable of processing numbers in integer (fixed-point) form only. Many games are written in Integer BASIC because its instructions can be executed very quickly. Compare **Applesoft BASIC.**

integrated circuit: An electronic circuit—including components and interconnections—entirely contained in a single piece of semiconducting material, usually silicon. Often referred to as an *IC* or a *chip*.

interface: The hardware or software necessary to allow two or more different pieces of equipment or software to work together. An interface usually has both a hardware and a software portion, referred to as the hardware interface and the software interface for a given piece of equipment or software.

interface card: A peripheral card that implements a particular interface (such as a parallel or serial interface) by which the computer can communicate with a peripheral device such as a printer or modem.

interpreter: A language translator that reads a program instruction by instruction and immediately translates each instruction for the computer to carry out. Compare **compiler.**

interrupt: A temporary suspension in the execution of a program that allows the computer to perform some other task, typically in response to a signal from a peripheral device or other source external to the computer.

inverse video: The display of text on the computer's display screen in the form of dark dots on a light background, instead of the usual light dots on a dark background.

I/O: See **input/output.**

I/O device: Input/output device. A device that transfers information into or out of a computer. See also **input, output, peripheral device.**

I/O link: A fixed location that contains the address of an input/output subroutine in the computer's Monitor program.

IOU: See **input/output unit.**

IWM: "Integrated Woz Machine"; the custom chip that controls floppy-disk drives.

joystick: A peripheral device with a lever, typically used to move creatures and objects in game programs; a joystick can also used in applications such as computer-aided design and graphics programs.

KB: See **kilobyte.**

keyboard: The set of keys, similar to a typewriter keyboard, used for entering information into the computer.

keyboard input connector: The connector inside the Apple II family of computers by which the keyboard is connected to the computer.

kilobyte (KB): A unit of measurement consisting of 1024 (2^{10}) bytes. In this usage, *kilo* (from the Greek, meaning a thousand) stands for 1024. Thus, 64 KB equals 65,536 bytes.

KSW: The symbolic name of the location in the computer's memory where the standard input link to the keyboard is stored. *KSW* stands for *keyboard switch*.

language: See **programming language.**

language card: A peripheral card that, when placed in slot 0 of a 48 KB Apple II or Apple II Plus, gives the computer a total of 64 KB of memory. In the Apple IIc, the memory that uses address space from $D000 through $FFFF—which corresponds to the memory added by the language card—is sometimes also called the "language card."

language translator: A system program that reads another program written in a particular programming language and either executes it directly or converts it into some other language (such as machine language) for later execution. Interpreters, compilers, and assemblers are all language translators.

LC: See **Language Card.**

leading zero: A zero occurring at the beginning of a decimal number, deleted by most computing programs.

least significant bit: The rightmost bit of a binary number. The least significant bit contributes the smallest quantity to the value of the number. Compare **most significant bit.**

LIFO: Acronym for "last in, first out" order, which is a storage and retrieval method in which the last item stored is the first retrieved.

line: See **program line.**

line feed: An ASCII character (decimal 10) that ordinarily causes a printer or video display to advance to the next line.

line width: The number of characters that fit on a line on the screen or on a page.

list: To display on a monitor, or print on a printer, the contents of memory or of a file.

load: To transfer information from a peripheral storage medium (such as a disk) into main memory for use—for example, to transfer a program into memory for execution.

local: Connected to or close by the host system.

location: See **memory location.**

logic: (1) In microcomputers, a mathematical treatment of formal logic using a set of symbols to represent quantities and relationships that can be translated into switching circuits, or *gates*. AND, OR, and NOT are examples of logical gates. Each gate has two states, open or closed, allowing the application of binary numbers for solving problems. (2) The systematic scheme that defines the interactions of signals in the design of an automatic data processing system.

logical operator: An operator, such as AND, that combines logical values to produce a logical result, such as true or false; sometimes called a *Boolean operator*. Compare **arithmetic operator, relational operator.**

logic board: See **main logic board.**

loop: A section of a program that is executed repeatedly until a limit or condition is met, such as an index variable reaching a specified ending value. See **loop.**

loop variable: See **index variable.**

Lo-Res graphics: The lowest-resolution graphics mode on the Apple II family of computers, consisting of an array of blocks 48 rows high by 40 columns wide, with 16 available colors. Compare **Hi-Res graphics, Double Hi-Res graphics.**

low-level language: A programming language that is relatively close to the form the computer's processor can execute directly. One statement in a low-level language corresponds to a single machine-language instruction. Examples are 6502 machine language, 6502 assembly language, and 68000 machine and assembly languages. Compare **high-level language.**

low-order byte: The less significant half of a memory address or other 2-byte quantity. In the 6502 microprocessor used in the Apple II family of computers, the low-order byte of an address is usually stored first, and the high-order byte second. The opposite is true for Macintosh computers.

low-power Schottky (LS): A type of transistor-transistor logic (TTL) integrated circuit having lower power and higher speed than a conventional TTL integrated circuit; named for Walter Schottky (1886–1956), a semiconductor physicist.

LS: See **low-power Schottky.**

machine language: The form in which instructions to a computer are stored in memory for direct execution by the computer's processor. Each model of computer processor (such as the 6502 microprocessor used in the Apple II family of computers) has its own form of machine language.

mainframe computer: A central processing unit or computer that is larger and more powerful than a minicomputer or a personal computer (microcomputer). Frequently called simply a *mainframe* for short.

main logic board: A large circuit board that holds RAM, ROM, the microprocessor, custom integrated circuits, and other components that make the computer a computer.

main memory: The part of a computer's memory whose contents are directly accessible to the microprocessor; usually synonymous with random-access memory (RAM). Programs are loaded into main memory, and that's where the computer keeps information while you're working. Sometimes simply called *memory*. See also **read-only memory, read-write memory.**

MARK parity: A bit of value 1 appended to a binary number for transmission. The receiving device checks for errors by looking for this value on each character. Compare **even parity, odd parity, SPACE parity.**

megabyte: A unit of measurement equal to 1024 kilobytes, or 1,048,576 bytes; abbreviated MB.

memory: A hardware component of a computer system that can store information for later retrieval. See **main memory, random-access memory, read-only memory, read-write memory.**

memory bank: See **bank.**

memory location: A unit of main memory that is identified by an address and can hold a single item of information of a fixed size. In the Apple II–family of computers, a memory location holds one byte, or eight bits, of information.

memory management unit (MMU): A custom chip that contains most of the logic that controls memory addressing in the Apple IIc family.

memory-resident: (1) Stored permanently in memory as firmware (ROM). (2) Held continually in memory even while not in use. DOS is a memory-resident program.

menu: A list of choices presented by a program, from which you can select an action.

MHz: Megahertz; one million hertz. See **hertz.**

microcomputer: A computer, such as any of the Apple II or Macintosh computers, whose processor is a **microprocessor.**

microprocessor: A computer processor contained in a single integrated circuit, such as the 6502 or 65C02 microprocessor used in the Apple II family of computers and the 68000 microprocessor used in the Macintosh family. The microprocessor is the central processing unit (CPU) of the microcomputer.

microsecond: One millionth of a second. Abbreviated µs.

MIG: See **multidrive interface glue.**

millisecond: One thousandth of a second. Abbreviated ms.

MMU: See **Memory Management Unit.**

mode: A state of a computer or system that determines its behavior. A manner of operating.

modem: Short for *MOdulator/DEModulator*. A peripheral device that links your computer to other computers and information services using the telephone lines.

modifier key: A key (Command, Caps Lock, Control, Option, Shift) that generates no keyboard events of its own, but changes the meaning of other keys or mouse actions. Also called a *control key*.

modulate: To modify or alter a signal so as to transmit information. For example, conventional broadcast radio transmits sound by modulating the amplitude (amplitude modulation, or *AM*) or the frequency (frequency modulation, or *FM*) of a carrier signal.

monitor: See **video monitor.**

Monitor: See **System Monitor.**

most significant bit: The leftmost bit of a binary number. The most significant bit contributes the largest quantity to the value of the number. For example, in the binary number 10110 (decimal value 22), the leftmost bit has the decimal value 16 (2^4). Compare **least significant bit.**

mouse: A small device you move around on a flat surface next to your computer. The mouse controls a pointer on the screen whose movements correspond to those of the mouse. You use the pointer to select menu items, to move data, and to draw with in graphics programs.

mouse button: The button on the top of the mouse. In general, pressing the mouse button initiates some action on whatever is under the pointer, and releasing the button confirms the action.

multidrive interface glue (MIG): A custom integrated circuit that implements a RAM buffer and provides control signals for internal and external 3.5-inch disk drives in the Apple IIc Plus computer.

nanosecond: One billionth of a second. Abbreviated ns.

nested loop: A loop contained within the body of another loop and executed repeatedly during each pass through the outer loop. See also **loop.**

nested subroutine call: A call to a subroutine from within the body of another subroutine.

nibble: A unit of data equal to half a byte, or four bits. A nibble can hold any value from 0 to 15.

NOT: A unary logical operator that produces a true result if its operand is false, and a false result if its operand is true. Compare **AND, OR, XOR.**

NTSC: Abbreviation for *National Television Standards Committee*. The committee that defined the standard format used for transmitting broadcast video signals in the United States.

NTSC video: The standard video format defined by the NTSC. Also called *composite video* because it combines all the video information, including color, into a single signal.

object code: See **object program.**

object program: The translated form of a program produced by a language translator such as a compiler or assembler. Also called *object code*. Compare **source program.**

odd parity: In data transmission, the use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits an odd number; used as a means of error checking. Compare **even parity, MARK parity, SPACE parity.**

opcode: See **operation code.**

Open Apple key: See **Command key.**

operand: A value to which an operator is applied. The value on which an operation code operates. Compare **argument.**

operating system: A program that organizes the actions of the parts of the computer and its peripheral devices. See also **disk operating system.**

operation code: The part of a machine-language instruction that specifies the operation to be performed. Often called *opcode*.

operator: A symbol or sequence of characters, such as + or AND, specifying an operation to be performed on one or more values (the operands) to produce a result. See **arithmetic operator, binary operator, logical operator, relational operator, unary operator.**

option: (1) Something chosen or available as a choice; for instance, items in a menu. (2) An argument whose provision is optional.

OR: A logical operator that produces a true result if either or both of its operands are true, and a false result if both of its operands are false. Compare **XOR, AND, NOT.**

output: Information transferred from a computer to some external destination, such as the display screen, a disk drive, a printer, or a modem.

output routine: A machine-language routine that performs the sending of characters. The standard output routine sends characters to the screen. A different output routine might, for example, send them to a printer.

overflow: The condition that exists when an attempt is made to put more data into a given memory area than it can hold; for example, a computational result that exceeds the allowed range.

overrun: A condition that occurs when the processor does not retrieve a received character from the receive data register of the Asynchronous Communications Interface Adapter (ACIA) before the subsequent character arrives. The ACIA automatically sets bit 2 (OVR) of its status register; subsequent characters are lost. The receive data register contains the last valid data word received.

page: An area of main memory containing text or graphical information being displayed on the screen. A segment of main memory 256 bytes long and beginning at an address that is an even multiple of 256.

page zero: See **zero page.**

parallel interface: An interface in which several bits of information (typically 8 bits, or 1 byte) are transmitted simultaneously over different wires or channels. Compare **serial interface.**

parity: Sameness of level or count, usually the count of 1 bits in each character, used for error checking in data transmission. See **even parity, MARK parity, odd parity, SPACE parity.**

Pascal: A high-level programming language with statements that resemble English phrases. Pascal was designed to teach programming as a systematic approach to problem solving. Named after the philosopher and mathematician Blaise Pascal.

pass: A single execution of a loop.

PC: See **Program Counter.**

PC board: See **printed-circuit board.**

peek: To read information directly from a location in the computer's memory.

peripheral: (adj) At or outside the boundaries of the computer itself, either physically (as an external disk drive) or in a logical sense (as a peripheral card). (n) Short for *peripheral device.*

peripheral bus: The bus used for transmitting information between the computer and peripheral devices connected to the computer's expansion slots or ports.

peripheral card: A removable printed-circuit board that plugs into an expansion slot. Peripheral cards allow a computer to use peripheral devices or to perform some subsidiary or peripheral function.

peripheral device: A piece of hardware—such as a video monitor, disk drive, printer, or modem—used in conjunction with a computer and under the computer's control. Peripheral devices are often (but not necessarily) physically separate from the computer and connected to it by wires, cables, or some other form of interface.

peripheral slot: See **expansion slot.**

phase: (1) A stage in a periodic process. A point in a cycle. For example, the 6502 microprocessor uses a clock cycle consisting of two phases called ø0 and ø1. (2) The relationship between two periodic signals or processes.

PILOT: Acronym for *Programmed Inquiry, Learning, Or Teaching*. A high-level programming language designed for teachers and used to create computer-aided instruction (CAI) lessons that include color graphics, sound effects, lesson text, and answer checking. SuperPILOT is an enhanced version of the original Apple II PILOT programming language.

pipelining: A feature of a processor that enables it to begin fetching the next instruction before it has finished executing the current instruction. All else being equal, processors with this feature run faster than those without it.

pixel: Short for *picture element.* A point on the graphics screen; the visual representation of a bit on the screen (white if the bit is 0, black if it's 1). Also, a location in video memory that maps to a point on the graphics screen when the viewing window includes that location.

pointer: An address of a location in memory that contains an item of data or the starting point of a routine. A vector.

point of call: The point in a program from which a subroutine or function is called.

poke: To store information directly into a location in the computer's memory.

pop: To remove the top entry from a stack, moving the stack pointer to the entry below it. Synonymous with *pull*. Compare **push.**

port: The firmware and hardware that interfaces an external or internal peripheral device to the Apple IIc computer. A port in the Apple IIc computer corresponds to a slot for an expansion card plus the hardware and firmware on the card in other Apple II computers.

power supply: A circuit that draws electrical power from a power outlet and converts it to the kind of power the computer can use.

power supply case: The metal case inside most Apple II and Macintosh computers that houses the power supply. The Apple IIc Plus has an internal power supply case; earlier versions of the Apple IIc use an external power supply case.

PR#: An Applesoft BASIC command that sends output to a slot or a machine-language program. It specifies an output routine in the ROM on a peripheral card or in a machine-language routine in RAM by changing the address of the standard output routine used by the computer.

P register: An 8-bit register in the 65C02 microprocessor that contains 7 status flags. Some of the flags are controlled by the program, and some by the microprocessor. These flags can be tested (read) by various 65C02 instructions. The P register is also called the Processor Status register.

primary character set: The set of characters displayed on the screen of an Apple IIc family computer when the enhanced video firmware is not active. Compare with **alternate character set.**

printed-circuit board: A hardware component of a computer or other electronic device, consisting of a flat, rectangular piece of rigid material, commonly fiber glass, to which integrated circuits and other electronic components are connected.

procedure: In the Pascal and Logo programming languages, a set of instructions that work as a unit; approximately equivalent to the term *subroutine* in BASIC.

processor: The hardware component of a computer that performs the actual computation by directly executing instructions represented in machine language and stored in main memory. See also **microprocessor.**

Processor Status Register: See **P register.**

ProDOS: An Apple II operating system designed to support hard disk drives like the ProFile, as well as floppy disk storage devices. ProDOS stands for *Professional Disk Operating System*.

ProDOS command: Any one of the 28 commands recognized by ProDOS.

program: (n) A set of instructions describing actions for a computer to perform in order to accomplish some task, conforming to the rules and conventions of a particular programming language. (v) To write a program.

program counter (PC): A 16-bit register in the 65C02 microprocessor that contains the address of the next instruction to be executed.

program line: The basic unit of an Applesoft BASIC program, consisting of one or more statements separated by colons (:).

programming language: A set of symbols and associated rules or conventions for writing programs. BASIC, Logo, and Pascal are programming languages.

prompt: A message on the screen that tells you of some need for response or action. A prompt usually takes the form of a symbol, a message, a dialog box, or a menu of choices.

prompt character: A text character displayed on the screen, usually just to the left of a cursor, where your next action is expected. The prompt character often identifies the program or component of the system that's prompting you. For example, Applesoft BASIC uses a square bracket prompt character (]); Integer BASIC, an angle bracket (>); and the System Monitor program, an asterisk (*).

prompt line: A specific area on the display reserved for **prompts.**

protocol: A formal set of rules for sending and receiving data on a communication line.

Protocol Converter: The name used for the first version of the SmartPort firmware; the Protocol Converter was introduced with the UniDisk 3.5 Apple IIc. See **SmartPort.**

push: To add an entry to the top of a stack, moving the stack pointer to point to it. Compare **pop.**

QWERTY keyboard: See **Sholes keyboard.**

radio-frequency (RF) modulator: A device that makes a television set work as a monitor.

RAM: See **random-access memory.**

random-access memory (RAM): Memory in which information can be referred to in an arbitrary or random order. As an analogy, a book is a random-access storage device in that it can be opened and read at any point. RAM usually means the part of memory available for programs from a disk; the programs and other data are lost when the computer is turned off. A computer with 512 KB of RAM has 512 kilobytes available to the user. (Technically, the read-only memory [ROM] is also *random access,* and what's called RAM should correctly be termed *read-write memory.*) Compare **read-only memory, read-write memory.**

random-access text file: A text file that is partitioned into an unlimited number of uniform-length compartments called *records.* When you open a random-access text file for the first time, you must specify its record length. No record is placed in the file until written to. Each record can be individually read from or written to—hence, *random-access.*

read: To transfer information into the computer's memory from outside the computer (such as a disk drive or modem) or into the computer's processor from a source external to the processor (such as the keyboard or main memory).

read-only memory (ROM): Memory whose contents can be read, but not changed; used for storing firmware. Information is placed into read-only memory once, during manufacture; it then remains there permanently, even when the computer's power is turned off. Compare **random-access memory, read-write memory.**

read-write memory: Memory whose contents can be both read and changed (or *written to*). The information contained in read-write memory is erased when the computer's power is turned off and is permanently lost unless it has been saved on a disk or other storage device. Compare **random-access memory, read-only memory.**

real number: In computer usage, a number that may include a fractional part; represented inside the computer in floating-point form. Because a real number is of infinite precision, this representation is usually approximate. Compare **integer.**

receive data register: A read-only register in the serial port ACIA (at $C098 for port 1 and $C0A8 for port 2) that stores the most recent character successfully received.

register: A location in a processor or other chip where an item of information is held and modified under program control. An area of memory used for a similar purpose.

relational operator: An operator, such as > (greater than), that operates on numeric values to produce a logical result. Compare **arithmetic operator, logical operator.**

Request-To-Send: An RS-232-C signal from a DTE to a DCE that serves to prepare the DCE for data transmission.

reserved word: A word or sequence of characters reserved by a programming language for some special use and therefore unavailable as a variable name in a program.

resident: See **memory-resident, disk-resident.**

return address: The point in a program to which control returns on completion of a subroutine or function.

RF modulator: See **radio-frequency modulator.**

RGB monitor: A type of color monitor that receives separate signals for each color (red, green, and blue).

ROM: See **read-only memory.**

routine: A part of a program that accomplishes some task subordinate to the overall task of the program.

row: A horizontal arrangement of character cells or graphics pixels on the screen.

RS-232 cable: Any cable that is wired in accordance with the RS-232 standard, which is the common serial data communication interface standard.

RTS: See **Request-To-Send.**

run: (1) To execute a program. When a program *runs,* the computer performs the instructions. (2) To load a program into main memory from a peripheral storage medium, such as a disk, and execute it.

save: To store information by transferring the information from main memory to a disk. Work not saved disappears when you turn off the computer or when the power is interrupted.

screen: See **display screen.**

screen hole: An address in memory that is within the range assigned to video dislplay, but that is not actually used by the video display. Screen holes are used by the I/O firmware in a variety of ways.

scroll: To move all the text on the screen upward or downward, and, in some cases, sideways.

serial interface: An interface in which information is transmitted sequentially, a bit at a time, over a single wire or channel. Compare **parallel interface.**

setup time: The amount of time a signal must be valid in advance of some event. Compare **hold time.**

Sholes keyboard: The standard layout of keys on a typewriter keyboard; its name is formed from the first six letters on the top row of letter keys. Also called a *QWERTY keyboard*. Compare **Dvorak keyboard**

silicon (Si): A solid, crystalline chemical element from which integrated circuits are made. Silicon is a *semiconductor;* that is, it conducts electricity better than insulators, but not as well as metallic conductors. Silicon should not be confused with silica—that is, silicon dioxide, such as quartz, opal, or sand—or with silicone, any of a group of organic compounds containing silicon.

simple variable: A variable that is not an element of an array.

68000: The microprocessor used in the Macintosh and Macintosh Plus.

6502: The microprocessor used in the Apple II, in the Apple II Plus, and in early models of the Apple IIe.

65C02: The microprocessor used in the enhanced Apple IIe, the extended keyboard Apple IIe, and the Apple IIc family of computers.

slot: A long, narrow socket inside the computer where you can install peripheral cards. Also called an *expansion slot.*

SmartPort firmware: A set of machine language routines in all Apple IIc-family computers except the original Apple IIc. The SmartPort firmware is used for performing block device I/O. The version of the SmartPort firmware used in the UniDisk 3.5 Apple IIc computer was called the *Protocol Converter*. The SmartPort firmware and the Protocol Converter are functionally identical.

soft switch: Also called a *software switch;* a means of changing some feature of the computer from within a program. A soft switch is a location in memory that produces some special effect whenever its contents are read or written.

software: A collective term for programs, the instructions that tell the computer what to do. They're usually stored on disks. Compare **hardware, firmware.**

source code: See **source program.**

source program: The form of a program given to a language translator, such as a compiler or assembler, for conversion into another form; sometimes called *source code*. Compare **object program.**

space character: A text character whose printed representation is a blank space, typed from the keyboard by pressing the Space bar.

SPACE parity: A bit value of 0 appended to a binary number for transmission. The receving device can look for this value on each character as a means of error checking. Compare **even parity, MARK parity, odd parity.**

S register: See **stack pointer.**

stack: (1) A list in which entries are added (pushed) or removed (popped) at one end only (the top of the stack), causing them to be removed in last-in, first-out (LIFO) order. (2) The portion of memory in page $01 used as a stack for 65C02 programs. Certain 65C02 instructions push bytes of data onto the stack or pull bytes of data from the stack.

stack pointer: An 8-bit register in the 65C02 microprocessor that points to the next byte of data to be used for the microprocessor's stack.

standard input: Data input by calls to the firmware routine KeyIn.

standard output: Data output by calls to the firmware routine COut.

start bit: A transition from a MARK signal to a SPACE signal for one bit-time, indicating that next string of bits represents a character.

starting value: The value assigned to the index variable on the first pass through a loop.

start up: To get the system running. Starting up is the process of first reading the operating system program from the disk, and then running an application program. Starting up is often called *booting*.

startup disk: A disk with all the necessary program files to set the computer into operation. Sometimes called a *boot disk*.

statement: A unit of a program in a high-level language that specifies an action for the computer to perform. A statement typically corresponds to several instructions of machine language.

status register: A location in the ACIA (at $C099 for port 1 and $C0A9 for port 2) that stores the state of two RS-232-C signals and the state of the transmit and receive data registers, as well as the outcome of the most recent character transfer.

step value: The amount by which the index variable changes on each pass through a loop.

stop bit: A MARK signal following a data string (or the optional parity bit), indicating the end of a character.

string: An item of information consisting of a sequence of text characters.

strobe: A signal whose change is used to trigger some action.

subroutine: A part of a program that can be executed on request from another point in the program and that returns control, on completion, to the point of the request.

Super Serial Card: A peripheral interface card used with the Apple IIe computer. The Super Serial Card interfaces perpheral devices—such as printers and modems—that use a serial data stream for communications. The serial ports in the Apple IIc family use a protocol very similar to that used by the Super Serial Card.

synchronous: A mode of data transmission in which a constant time interval exists between transmission of successive bits, characters, or events. Compare **asynchronous.**

synchronous transmission: A transmission process that uses a clocking signal to ensure an integral number of unit (time) intervals between any two characters. Compare **asynchronous transmission.**

syntax: (1) The rules governing the structure of statements or instructions in a programming language. (2) A representation of a command that specifies all the possible forms the command can take.

system: A coordinated collection of interrelated and interacting parts organized to perform some function or achieve some purpose—for example, a computer system comprising a processor, keyboard, monitor, and disk drive.

system configuration: See **configuration.**

System Monitor: A system program built into the firmware of the Apple IIc, used for directly inspecting or changing the contents of main memory and for operating the computer at the machine-language level.

system program: A program that makes the resources and capabilities of the computer available for general purposes, such as an operating system or a language translator. Compare **application program.**

system software: The component of a computer system that supports application programs by managing system resources such as memory and I/O devices.

tab: An ASCII character that commands a device such as a printer to start printing at a preset location (called a *tab stop*). There are two such characters: horizontal tab (HT, $09) and vertical tab (VT, $0B).

television set: A display device capable of receiving broadcast video signals (such as commercial television broadcasts) by means of an antenna. Can be used in combination with a radio-frequency modulator as a display device for the Apple II family of computers. Compare **video monitor.**

text: (1) Information presented in the form of readable characters. (2) The display of characters on a display screen. Compare **graphics.**

text window: An area on the video display screen within which text is displayed and scrolled.

timing generator (TMG): The custom integrated circuit that generates clock and timing signals used by the Apple IIc.

TMG: See **timing generator.**

traces: Electrical paths that connect the components on a circuit board.

transistor-transistor logic (TTL): (1) A family of integrated circuits having bipolar circuit logic; TTLs are used in computers and related devices. (2) A standard for interconnecting such circuits, which defines the voltages used to represent logical zeros and ones.

transmit data register: A location in the ACIA (at location $C098 for port 1 and $C0A8 for port 2) that holds the current character to be transmitted.

troubleshoot: To locate and correct the cause of a problem or malfunction, especially in hardware. Compare **debug.**

TTL: See **transistor-transistor logic.**

unary operator: An operator that applies to a single operand. For example, the minus sign (–) in a negative number such as –6 is a unary arithmetic operator. Compare **binary operator.**

unconditional branch: A branch that does not depend on the truth of any condition. Compare **conditional branch.**

value: An item of information, such as a number or a string, that can be stored in a variable.

variable: (1) A location in the computer's memory where a value can be stored. (2) The symbol used in a program to represent such a location. Compare **constant.**

VBL: Mnemonic for *vertical blanking*. See **vertical blanking signal, vertical blanking interrupt.**

vector: (1) An address of a location in memory, often the starting address or entry point for a special routine. Application programs and operating systems use vectors to get to an address quickly. (2) A memory location used to hold a vector, or the address of such a location.

vertical blanking interrupt: An interrupt to the CPU syncharonized with the vertical blanking signal. See also **vertical blanking signal.**

vertical blanking signal: A signal used to synchronize the turning off (blanking) of the electron beam in the video display with the return of the beam from the bottom to the top of the screen. In the Apple IIc–family computers, the vertical blanking signal is asserted 60 times each second. See also **vertical blanking interrupt.**

video: (1) A medium for transmitting information in the form of images to be displayed on the screen of a cathode-ray tube. (2) Information organized or transmitted in video form.

video monitor: A display device that can receive video signals by direct connection only, and that cannot receive broadcast signals such as commercial television. Can be connected directly to the computer as a display device. Compare **television set.**

viewport: All or part of the display screen used by an application program to display a portion of the information (such as a document, picture, or worksheet) on which a program is working. Compare **window.**

volume: A general term referring to a storage device; a source of or a destination for information. A volume has a name and a volume directory with the same name. Its information is organized into *files*.

warm start: The process of putting the Apple IIc into a known state. Depending on what operating system and application program, if any, are running when the warm start takes place, the computer may wind up back in the operating sytem or at the startup point for an application program. Compare **cold start.**

window: The portion of a collection of information (such as a document, picture, or worksheet) that is visible in a viewport on the display screen. Compare **viewport.**

word: A group of bits that is treated as a unit; the number of bits in a word is a characteristic of each particular computer. In the Apple IIc family of computers, a word is 16 bits (2 bytes) long.

write: To transfer information from the computer to a destination external to the computer (such as a disk drive, printer, or modem) or from the computer's processor to a destination external to the processor (such as main memory).

write-enable notch: The square cutout on one edge of a 5.25-inch disk's jacket. If there is no write-enable notch, or if it is covered with a write-protect tab, the disk drive can read information from the disk, but cannot write on it.

XOR: Exclusive-OR. A Boolean (logical) operation that produces a true (1) result if *one and only one* of its inputs is true (1). Compare **OR, AND,** and **NOT.**

X register: One of the two index registers in the 6502 microprocessor.

Y register: One of the two index registers in the 6502 microprocessor.

zero page: The first page (256 bytes) of memory in the Apple II family of computers, also called *page zero*. Since the high-order byte of any address in this page is zero, only the low-order byte is needed to specify a zero-page address; this makes zero-page locations more efficient to address, in both time and space, than locations in any other page of memory.

Index

W

X

Y

Z

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh® computers and Microsoft® Word. Proof pages were created on the Apple LaserWriter® Plus. Final pages were created on the Varityper® VT600™. POSTSCRIPT®, the LaserWriter page-description language, was developed by Adobe Systems Incorporated. Some of the illustrations were created using Adobe Illustrator™.

Text type is ITC Garamond® (a downloadable font distributed by Adobe Systems). Display type is ITC Avant Garde Gothic®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier, a fixed-width font.